Universität Hildesheim
Fachbereich III – Informations- und Kommunikationswissenschaften
Institut für Informatik
Gruppe Intelligente Informationssysteme

Masterarbeit
zur Erlangung des akademischen Grades eines
Master of Science Informationstechnologie
im Weiterbildungsstudiengang Informationstechnologie

Integration und Modellierung von verteilten Geschäftsprozessen mittels Web Services am Beispiel eines Prozessportals

---

1. Gutachter: Prof. Dr. Klaus-Dieter Althoff
2. Gutachter: Dipl.-Inform. Martin Schaaf

erstellt bei:
abaXX Technology AG
Forststraße 7
70174 Stuttgart

eingereicht von:

Joachim Pfister
(Joachim-Pfister@gmx.de)

Burladingen, im März 2006

Abstract

Die vorliegende Arbeit untersucht Möglichkeiten zur Integration und Modellierung von verteilten Geschäftsprozessen unter Verwendung von Web Services. Neben den Grundlagen werden weiterführende Konzepte im Bereich Web Services und Service-oriented Architectures thematisiert, wie z.B. Sicherheitsaspekte, Transaktionen und die Orchestrierung und Choreographie von Services. Als Anwendungskontext dient die Prozessportal-Lösung der Firma abaXX Technology AG (Stuttgart), innerhalb welcher Web Services durch eine im Rahmen dieser Arbeit erstellte, prototypische Software („Web Service Import Wizard“) nutzbar gemacht werden.

Schlagworte: Web Services, verteilte Geschäftsprozesse, Prozessportal, Integration, Orchestrierung

---

The integration and modelling of distributed business processes using Web Services is the subject of this Master's thesis. Fundamental standards and principles of Web Services are illustrated as well as existing and forthcoming extensions, e.g. security, transactions, orchestration and choreography of Web Services. This Master's thesis is embedded in the context of a process portal solution developed by abaXX Technology AG (Stuttgart, Germany). Within this application context, a prototype ("Web Service Import Wizard“) was developed to integrate and consume Web Services in a process model.

Key words: Web Services, distributed business processes, process portal, integration, orchestration

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

2PC	Two-Phase-Commit
AXIS	Apache eXtensible Interaction System
API	Application Programming Interface
BPEL	Business Process Execution Language
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Common Object Model
EAI	Enterprise Application Integration
ESB	Enterprise Service Bus
HTTP	Hyper Text Transport Protocol
JFC	Java Foundation Classes
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
J2EE	Java 2 Enterprise Edition
JMS	Java Messaging Service
MEP	Message Exchange Patterns
OASIS	Organization for the Advancement of Structured Information Standards
ORB	Object Request Broker
OWL	Web Ontology Language
QoS	Quality of Service
RDF	Resource Description Framework
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SMTP	Simple Mail Transfer Protocol
SOA	Service-Oriented Architecture
SOAP	(ehemals) Simple Object Access Protocol
SWT	Standard Widget Toolkit
UBR	UDDI Business Registry
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WfMC	Workflow Management Coalition
WS	Web Service
WSFL	Web Service Flow Language
WSIF	Web Services Invocation Framework
WSDL	Web Service Description Language
WSRP	Web Services for Remote Portlets
WS-I	Web Service Interoperability Organization
XML	eXtensible Markup Language

1 Einleitung

> „By 2003, more than 40 percent of all Internet-oriented interactions will leverage Web components from multiple enterprises, and, by 2004, Web services will dominate deployment of new application solutions for Fortune 2000 companies."
>
> *(Gartner 2002)*

1.1 Einleitung und Motivation

Kaum ein Anwender bzw. ein Hersteller von Software hätte gedacht, dass mit der Standardisierung von SOAP und WSDL im Jahre 2001 der Startschuss für einen Hype rund um die Thematik Web Services und Service-oriented Architecture erteilt wurde. An diese neuen, standardisierten Technologien wurden (und werden immer noch) große Erwartungen geknüpft – beispielsweise geschürt durch Analysen von IT-Beratungsunternehmen, wie das eingangs erwähnte Zitat zeigt.

Mittlerweile ist ein wenig Ruhe in die Thematik Web Services eingekehrt. Unrealistische Erwartungen wichen einer pragmatischeren Sichtweise auf diese Technologien. Bei Web Services ist inzwischen ein Reifegrad der einzelnen Technologiekomponenten erreicht, der einen produktiven Einsatz in verschiedenen Szenarien überwiegend problemlos ermöglicht und angesichts bestimmter Architektur-Paradigmen sogar erforderlich macht (z.B. wenn eine Service-oriented Architecture umgesetzt werden soll).

Web Services sind heute ein gutes Instrument, um plattformübergreifend Anwendungen miteinander zu koppeln. Über genau definierte Schnittstellen können Daten interoperabel, z.B. zwischen einer Java-Anwendung, die auf einer Sun-Plattform läuft und einer Microsoft C#-Anwendung auf einer Intel-Plattform, ausgetauscht werden. Die eigentliche Implementierung eines Dienstes ist dabei nicht von Interesse, da die Anknüpfungspunkte über wohldefinierte Schnittstellen gekapselt sind und der Datenaustausch über etablierte Standards abläuft.

Die alleinige Auseinandersetzung mit den Technologien und Standards zum Thema „Web Services" hat – obgleich ihrer Bedeutung – eher den Charakter einer theoretischen Fingerübung. Aus diesem Grund wurde die Nutzung von Web Services in den Kontext eines realen Anwendungsszenarios gestellt. Der Anwendungskontext ist im Falle dieser Arbeit durch die Prozessportal-Lösung der Firma abaXX Technology gegeben. Mit dieser Software-Lösung können komplexe Anwendungen und Abläufe innerhalb einer Portal-Anwendung gestaltet werden. Innerhalb der Abarbeitung eines Prozessmodells sollen

Web Services genutzt werden, um z.B. Informationen aus einer anderen Anwendung auf einfache Weise zu integrieren. Zu diesem Zweck wurde im Rahmen des praktischen Teils dieser Arbeit eine prototypische Software entwickelt („Web Service Import Wizard“), die das Einbinden von Web Services in ein Prozessmodell zum Modellierungszeitpunkt für den Entwickler vereinfachen soll.

Daher werden in dieser Arbeit die Prinzipien und Standards im Bereich von Service-oriented Architectures mit Web Services vorgestellt, um, basierend auf diesen Grundlagen, das Vorgehen bei der Implementierung des „Web Service Import Wizards“ zu beschreiben.

1.2 Aufbau der Arbeit

Die Einbettung dieser Arbeit in einen anwendungsbezogenen Kontext geschieht in *Kapitel 2*. Zuerst werden die Grundlagen einer Portalsoftware, von Geschäftsprozessen und von Workflow-Management-Systemen erläutert. Weiterhin wird die Prozessportal-Lösung der Firma abaXX Technology vorgestellt um daran die Ziele und Anforderungen für den praktischen Teil dieser Arbeit abzuleiten: Die Erstellung eines „Web Service Import Wizards“ zum Einbinden von Web Services in die Prozessportal-Lösung. Kapitel drei und vier beschäftigen sich mit den Grundlagen zur Nutzung von Web Services, während in Kapitel fünf und sechs weiterführende Konzepte aufgezeigt werden.

In *Kapitel 3* werden die Grundlagen einer Service-oriented Architecture beschrieben, die mittels Web Services umgesetzt werden kann. Eine Einordnung von Web Services in verwandte Technologien, eine Beschreibung ihrer Vorteile und die Entwicklung hin zu einem Enterprise Service Bus sind Gegenstand dieses Kapitels.

Eine Beschreibung der benötigten Technologien zur Nutzung von Web Services findet sich in *Kapitel 4*. Das Kommunikationsprotokoll sowie Standards zur Dienstbeschreibung und zur Dienstsuche werden dargestellt.

Kapitel 5 beschäftigt sich mit den Möglichkeiten zur Modellierung von Geschäftsprozessen. Es wird auf die Unterschiede zwischen Orchestrierung und Choreographie von Web Services eingegangen, wobei schwerpunktmäßig die Sprache WS-BPEL zur Modellierung von Geschäftsprozessen beschrieben wird.

Eine Darstellung von Web Service-Technologien zu den Themen Sicherheit und Transaktionen erfolgt in *Kapitel 6*. Dort wird ebenfalls auf die Bedeutung dieser Technologiekomponenten für einen überbetrieblichen Einsatz hingewiesen.

In *Kapitel 7* werden Gründe und Grundlagen für den Einsatz von Web Service Frameworks aufgezeigt. Drei verschiedene Frameworks werden vorgestellt, um deren Vorteile und Nachteile einander gegenüberzustellen.

Das Vorgehen im praktischen Teil dieser Arbeit wird in *Kapitel 8* beschrieben. Zunächst werden, ausgehend von den Zielen und der Anforderungsanalyse in Kapitel 2, Anwendungsfälle definiert. Die Auswahl eines Web Service Frameworks, die im vorangegangenen Kapitel vorgestellt wurden, wird in diesem Kapitel begründet. Das weitere Vorgehen bei der Implementierung, die eingesetzten Werkzeuge sowie eine Beschreibung der prototypischen Lösung ist ebenso Bestandteil dieses Kapitels. Bei der Implementierung auftretende Fragestellungen von besonderer Bedeutung wurden als eigenständiges Kapitel herausgearbeitet.

Das letzte *Kapitel 9* beendet die Arbeit mit einem Ausblick hinsichtlich der Weiterentwicklung von Web Services und deren Verknüpfung mit Techniken des Semantic Webs und den generellen Stärken von Web Services bei der Anwendungsintegration. Zudem werden Erweiterungsmöglichkeiten aufgezeigt, die sich beim Einsatz von Web Services im Kontext des Prozessportals von abaXX Technology ergeben könnten.

2 Die Prozessportal-Lösung von abaXX Technology als Anwendungskontext

Dieses Kapitel widmet sich der Beschreibung der grundlegenden Architektur einer Portalsoftware und deren konkreter Ausgestaltung anhand der Portallösung der Firma abaXX Technology, die den Anwendungskontext dieser Arbeit bildet. Weiterhin erfolgt eine Beschreibung der Analyse und Entwurfsphase, in der die Anforderungen an und die Ziele der im Zuge dieser Arbeit erstellten prototypischen Software vorgestellt werden.

2.1 Allgemeine Entwurfsprinzipien von Portalen

Der Begriff Portal (lat. *porta* = [Stadt-]Tor, Pforte, Eingang, Zugang; „prunkvolles Tor, architektonisch besonders gestalteter Haupteingang“ (Duden 2001, 619)) wurde allgemein auf „Einstiegsseiten“ im Internet übertragen. Vorgänger waren beispielsweise Unternehmenswebsites oder das Intra- oder Extranet von Unternehmen. Vlachikis et al. (2005, 11) definieren ein **Unternehmensportal** „[...] als eine Applikation, welche basierend auf Webtechnologien einen zentralen Zugriff auf personalisierte Inhalte sowie bedarfsgerecht auf Prozesse bereitstellt.“

Ein Portal soll außerdem die Arbeit von heterogenen Gruppen unterstützen, was z.B. technisch durch ein Verknüpfen heterogener Anwendungen über eine zentrale Plattform unter einer einheitlichen Benutzeroberfläche geschehen kann. Zu den Kernmerkmalen der Portalfunktionalität zählt neben der Informationsverbreitung die Prozessorientierung (vgl. Vlachikis et al. 2005, 11).

Unternehmensportale lassen sich anhand ihrer Zielgruppe in vier Kategorien einteilen (vgl. Vlachikis et al. 2005, 12f.):

- **Mitarbeiterportale** als Schnittstelle zwischen Mitarbeitern und den Prozessen und Systemen eines Unternehmens,
- **Geschäftskundenportale** zur Unterstützung zwischenbetrieblicher Prozesse, z.B. im Rahmen von Marketing, Vertrieb oder Service,
- **Lieferantenportale** als Grundlage zur Einbeziehung von Lieferanten durch z.B. Prozesse zur Angebotsabgabe und Rechnungsstellung,
- **Endkundenportale** als Anlaufstelle für Marketing-, Vertriebs- und Serviceprozesse für Endkunden.

Das Fraunhofer Institut für Arbeitswirtschaft und Organisation (IAO) entwickelte eine dreischichtige Referenzarchitektur (mittlerweile in der Version 2.0, Stand Januar 2005), die die Funktionalitäten von Portalsoftware herstellerunabhängig aufzeigt (siehe Abbildung 2.1).

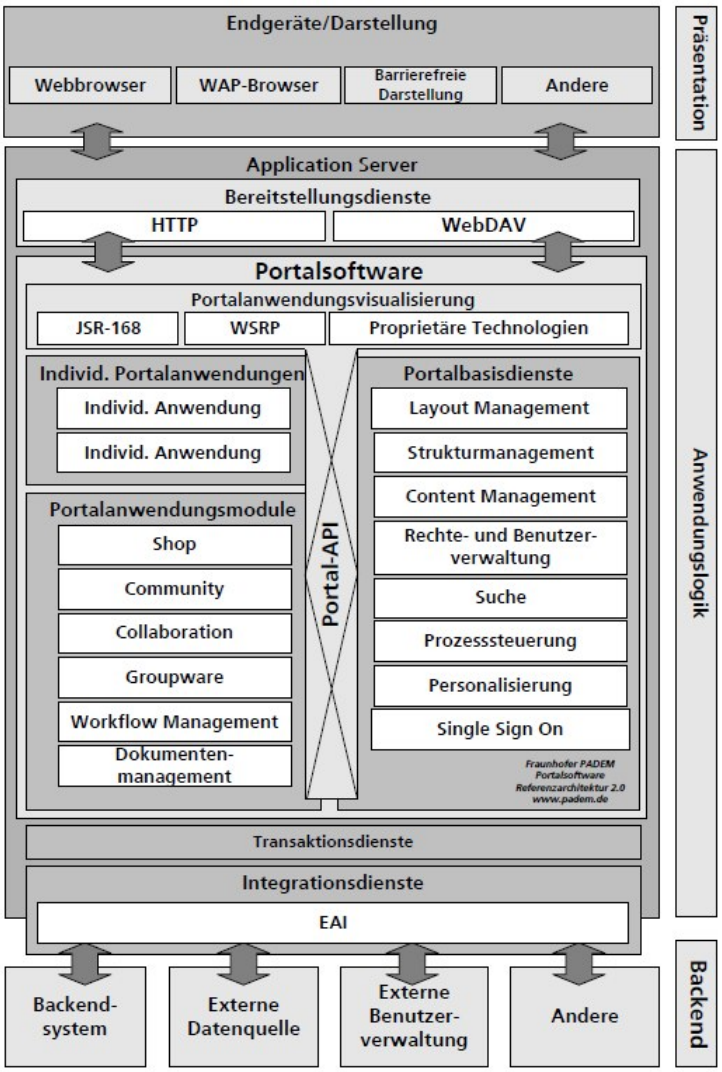

Abbildung 2.1: Referenzarchitektur für Portalsoftware (Vlachikis et al. 2005, 15)

In der **Präsentationsschicht** liegen die Endgeräte, die zur Darstellung der Portalinhalte verwendet werden. Die **Anwendungslogik** ist das Herzstück der Portalsoftware, die auf einem Application Server basiert. Die Portalbasisdienste der Portalsoftware stellen grundlegende Funktionen für die Erstellung und den Betrieb eines Portals bereit (Teilkomponenten siehe Vlachikis et al. 2005, 30ff.). Die Portalanwendungsmodule sind meist optional mit der Portalsoftware ausgelieferte Komponenten. Die Schicht des sog. **Backends** wird zur Anbindung an andere betriebliche Informationssysteme genutzt (z.B. „Enterprise Resource Planning"-Systeme wie SAP). (vgl. Vlachikis et al. 2005, 16f.)

2.2 Geschäftsprozesse und Workflows

„Ein **Geschäftsprozess** ist eine Folge von Aktivitäten, die in einem logischen Zusammenhang stehen, inhaltlich abgeschlossen sind und unter Zuhilfenahme von Ressourcen und eingehenden Informationen durch Menschen und/oder Maschinen auf ein Unternehmensziel hin ausgeführt werden. Ein **Workflow** ist die informationstechnische Realisierung eines Geschäftsprozesses.“ (Martens 2004, 12)

Ein Geschäftsprozess wird durch Eintreten eines Geschäftsvorfalls initiiert, z.B. durch den Eingang einer Bestellung eines Kunden. Bei der Bearbeitung des Geschäftsvorfalls durchläuft dieser verschiedene Stationen und Zustände, bis er ganz abgearbeitet ist und im Normalfall eine vollständige Befriedigung der Kundenwünsche eingetreten ist. Zur Klassifikation von Geschäftsprozessen zieht Martens als Merkmale zum einen das wirtschaftliche Gewicht (Kernprozess bei einer Versicherung ist der Abschluss einer Versicherungspolice; die Spesenabrechnung der Mitarbeiter ist eher ein untergeordneter Geschäftsprozess) und zum anderen die Ausführungshäufigkeit heran (z.B. Einberufung einer Hauptversammlung einer Aktiengesellschaft als seltenerer – wichtiger – Geschäftsprozess; Bearbeitung von Spesenabrechnungen ein häufiger stattfindender Geschäftsprozess). „Häufig ausgeführte und wirtschaftlich gewichtige Geschäftsprozesse sind eher stark strukturiert und präzise organisiert. Diese Prozesse bezeichnet man auch als *Production Workflow*“ (Martens 2004, 12f.).

2.3 Grundlagen des Workflow-Managements

Mit dem Begriff *Workflow-Management* werden Aktivitäten zusammengefasst, die in Zusammenhang mit der informationstechnischen Unterstützung von Geschäftsprozessen stehen. Martens (2004, 13) zählt dazu:

- „das Planen und Spezifizieren (Geschäftsprozessmodellierung),
- das Analysieren und Optimieren (Business Process Reengineering),
- das Definieren und Präzisieren (Workflow-Modellierung),
- das Ausführen und Administrieren (Workflow-Enactment) sowie
- das Überwachen und Auswerten (Workflow-Monitoring) mit Hilfe eines oder mehrerer computer-gestützter Werkzeuge.“

Die meisten Ansätze im Workflow-Management stammen aus dem graphentheoretischen Bereich, z.B. in Form einer Modellierung durch Petri-Netze. Im Jahre 1993 wurde die Workflow Management Coalition (WfMC) gegründet[1]. Sie hat als internationale, herstellerübergreifende Organisation zum Ziel, durch Erarbeiten von Standards ein Vereinheitlichen der Terminologie, der Interoperabilität und des Datenaustausches zwischen Workflow-Produkten zu gewährleisten.

[1] `http://www.wfmc.org`, verifiziert am 17.01.2006, 10:54 MEZ

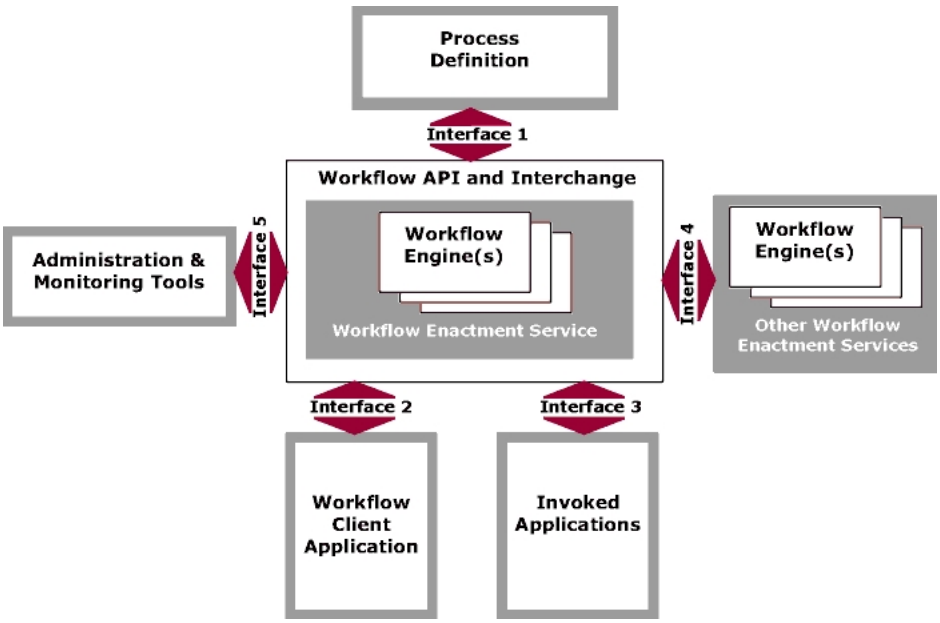

Abbildung 2.2: Workflow-Referenz-Modell der WfMC (Quelle: `http://www.wfmc.org/images/model-1.gif`, verifiziert am 17.01.2006, 13:51 MEZ)

Das Workflow-Referenzmodell (siehe Abbildung 2.2) stellt ein Architekturmodell dar, das „die Komponenten und deren Schnittstellen untereinander definiert, um eine weitgehende Systemunabhängigkeit und Interoperabilität zu erreichen." (Dostal et al. 2005, 199) Die einzelnen Komponenten sind (vgl. Dostal et al. 2005, 199f.):

- **Process Definition Tools Interface (1):** Mittels dieser Komponente wird der Prozess definiert, z.B. mittels graphischen Tools, die eine XML-Datei erstellen.
- **Workflow Client Application Interface (2):** Zusammenstellung aller Schnittstellen, damit eine Software zur Modellierung eines Workflows mit dem Workflow-Managementsystem interagieren kann.
- **Workflow Engine(s):** Die standardisierte Prozessdefinition wird von einer Anwendung, der sog. „Workflow-Engine", ausgeführt. Dies wird in der WfMC-Terminologie mit „Workflow Enactment Service" bezeichnet.
- **Invoked Application Interface (3):** Beinhaltet Schnittstellen, die zum Ansprechen von externen Anwendungen aus Workflow-Aktivitäten heraus genutzt werden können. Diese Schnittstellen ermöglichen eine unabhängige Definition vom verwendeten Zielsystem.
- **Other Workflow Engines / Workflow Interoperability Interface (4):** Schnittstellen zur Interaktion mit anderen Workflow-Systemen, um ein größeres Workflow-Szenario zusammenzustellen. Hierbei liegt ein bereits definierter Satz von Diensten vor.
- **Administration & Monitoring Tools Interface (5):** Diese Schnittstellen sorgen für das Protokollieren von Workflow-Aktivitäten (z.B. auf Grund gesetzlicher Vorgaben).

Die Elemente der Workflow-Referenzarchitektur spiegeln sich in verschiedenen Web Service-Spezifikationen wieder, v.a. im Zusammenhang mit der Modellierung von Geschäftsprozessen.

2.4 Vorstellung der abaXX-Prozessportal-Lösung

Die Portallösung von abaXX Technology ist eine durchgängig in Java entwickelte, komponentenbasierte Portalsoftware. Diese Portalsoftware ist in verschiedenen Ausgaben mit unterschiedlichem Funktionsumfang erhältlich. Bezogen auf das zuvor benannte Referenzmodell des Fraunhofer IAO (vgl. Kapitel 2.1) können somit verschiedene Portalanwendungsmodule zu den Basisdienstleisungen des abaXX-Portals hinzugefügt werden, wie z.B. eine Workflow Management-Komponente (innerhalb der abaXX.portal process edition die abaXX process.component) oder eine Shop-Komponente (innerhalb der abaXX.portal commerce edition die abaXX commerce.component).

Mittels der Portallösung von abaXX lassen sich Portale für verschiedene Bedürfnisse und Zielgruppen entwickeln (vgl. Kapitel 2.1). Als Beispiel soll hierbei ein Mitarbeiterportal innerhalb eines Unternehmens angeführt werden. Mitarbeiterportale können zu einer effizienten Gestaltung von innerbetrieblichen Geschäftsprozessen beitragen, indem z.B. die Abwicklung eines Reiseantrags papierlos mit Hilfe des Prozessportals abgewickelt wird. Hierfür eignet sich die abaXX process.component mit ihrer Workflow-Komponente. Sie basiert auf den in Kapitel 2.3 vorgestellten Prinzipien der WfMC.

Zum graphischen Modellieren eines Geschäftsprozesses (in diesem Fall die Bearbeitung eines Reiseantrags) wird der *„Process Modeler“* eingesetzt (siehe Abbildung 2.3). Mittels Drag&Drop können aus der *„Activities Gallery“* (im rechten Teil in Abbildung 2.3) vorgefertigte Elemente zum Erstellen eines Prozessmodells verwendet werden, wie z.B. die „Decider“-Aktivität, die in Abbildung 2.3 als Raute mit der Beschriftung „Approved“ dargestellt wird. Dieser Activity Gallery können auch eigene Aktivitäten hinzugefügt werden. Die graphische Modellierung erlaubt eine einfache Anpassungen von Prozessen an sich ändernde Anforderungen. Durch die gemeinsame Nutzung eines graphischen Modellierungswerkzeugs wird zudem die Kommunikation zwischen der Fachabteilung, die die Prozesse entwirft, und den Software-Entwicklern, die die Prozesse programmtechnisch umsetzen, wesentlich verbessert.

Der erzeugte Prozess wird in einer Prozessdefinition in einer XML-Datei abgespeichert (process definition). Zur Laufzeit werden die Prozessdefinitionen geladen, geparst und in ausführbare Instanzen eines Prozesses überführt. Die Workflow-Engine sorgt für die Abarbeitung der Prozesse. Ein Prozess setzt sich aus mehreren Aktivitäten zusammen, die die kleinsten Elemente einer Prozessdefinition darstellen. Im Rahmen der Abarbeitung eines Prozesses werden die Aktivitäten-Objekte erzeugt, ausgeführt und ausgewertet. Der Datenaustausch erfolgt über den sog. *Prozesskontext* (process context), auf den alle

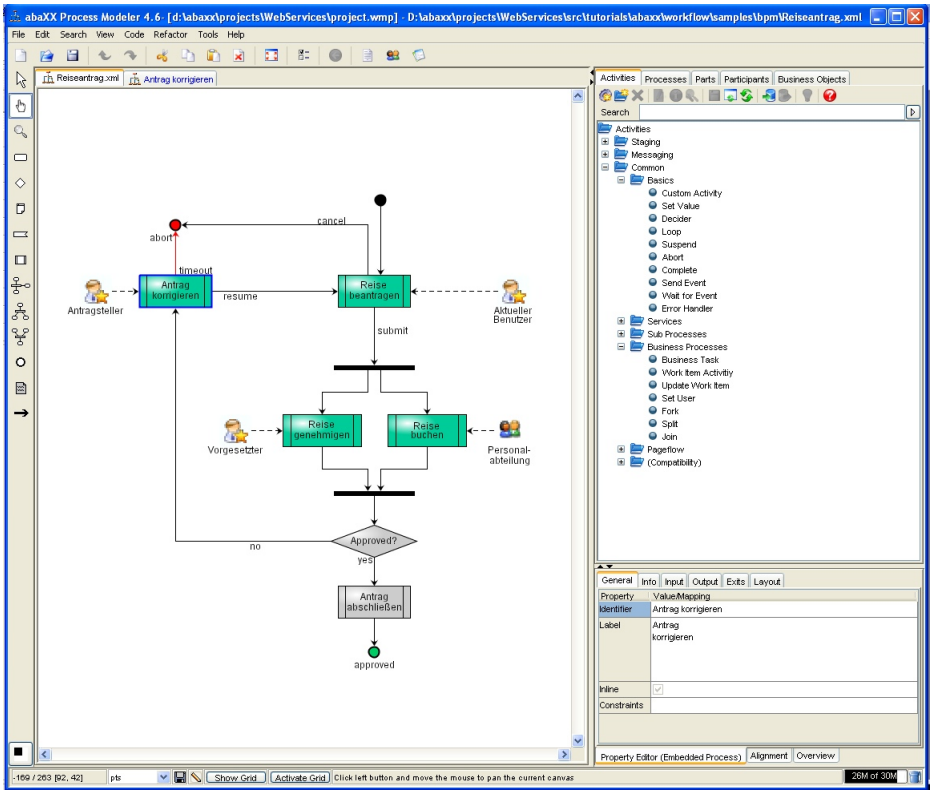

Abbildung 2.3: Modellierung eines Geschäftsprozesses mit dem abaXX Process Modeler

Aktivitäten eines Prozesses gemeinsam zugreifen können. Es handelt sich hierbei um einen Blackboard-Ansatz, der bei Alonso et al. (2004, 266) vorgestellt wird. Abbildung 2.4 stellt diese Konzepte in graphischer Form dar (vgl. abaXX 2005).

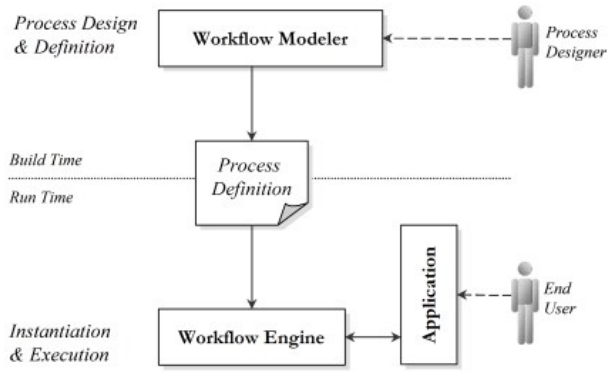

Abbildung 2.4: Workflow-Management mittels der abaXX.components (abaXX 2005)

Die Workflow-Engine unterscheidet zwischen drei Arten von Prozessen (vgl. abaXX 2005):

- **Business Processes:** Geschäftsprozesse sind langlaufende Prozesse, an denen mehrere Beteiligte eine Teilaufgabe erledigen. Sie dienen zur Beschreibung von realen Arbeitsabläufen innerhalb einer Organisation. Mit Hilfe von sog. *„Work Items“* wird ein Wechsel der Person modelliert, die eine Teilaufgabe eines Prozesses zu bearbeiten hat. Dabei wird der Prozess suspendiert (d.h. in Wartestellung versetzt) und

persistent abgespeichert. Der Prozess wird dann fortgeführt, sobald der Prozessbeteiligte in seiner *„Work List“* das entsprechende *„Work Item“* zur Weiterbearbeitung auswählt.

- **Page Flows:** Diese Art von Prozess beschreibt die Reihenfolge der Web-Seiten, die ein Nutzer während der Interaktion mit der Anwendung angezeigt bekommt. Im Gegensatz zu den Business Processes findet bei einem Page Flow kein Wechsel der beteiligten Personen statt. Zudem wird der Zustand eines Page Flows nicht dauerhaft gespeichert, um evtl. später wieder darauf zurück zu greifen.
- **Micro Flows:** Das sind Prozesse von sehr kurzer Dauer, bei denen keine Nutzerinteraktion erforderlich ist. Sie werden dazu eingesetzt, einzelne, synchrone Operationen unterbrechungsfrei ablaufen zu lassen.

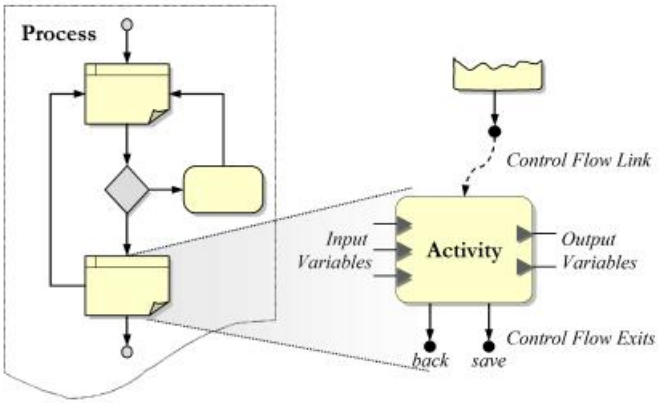

Abbildung 2.5: Prozessmodell der abaXX Workflow-Engine (abaXX 2005)

Eine *Aktivität* beinhaltet öffentliche Schnittstellen, um Ein- und Ausgabevariablen sowie Ausgänge für den Kontrollfluss festzulegen (siehe Abbildung 2.5). Sie werden in Form von *„Activity Implementations“* durch Java-Klassen definiert. *Prozesse* setzen sich aus den unterschiedlichen Aktivitäten zusammen. Sie können auch als Sub-Prozesse ineinander verschachtelt werden. Der *Kontrollfluss* steuert den Ablauf eines Prozesses und wird über die Prozessdefinition festgelegt. Dabei werden die Kontrollfluss-Ausgänge der Aktivitäten in der gewünschten Reihenfolge miteinander verknüpft. Der *Datenfluss* legt innerhalb einer Prozessdefinition fest, welche Eingabe- oder Ausgabevariablen den im Prozess vorhandenen Aktivitäten zugewiesen werden (vgl. abaXX 2005).

2.5 Ziele und Anforderungsanalyse

Der abaXX Process Modeler bietet die Möglichkeit, sog. *„Custom Activities“* zu implementieren. Das sind „leere“ Aktivitäten, deren Funktionalität durch den Programmierer zu erstellen ist. Diese Custom Activities werden in dieser Arbeit als Basis benutzt, um

aus einem Prozessmodell heraus „Web Services“ zu integrieren und anzusprechen. Beispielsweise können, um das Beispiel aus dem vorangegangenen Kapitel fortzusetzen, Web Services dazu genutzt werden, die Verfügbarkeit von Flügen oder Mietwagen zu überprüfen und evtl. sogar automatisch eine Buchung durchzuführen. Man kann sich Web Services – vereinfacht formuliert – als Dienste vorstellen, die von einem anderen Anbieter erbracht werden. Diese Dienste können in die eigene Infrastruktur und Anwendungen eingebunden werden. Eine genauere Definition von Web Services und den damit verbunden Konzepten erfolgt im sich anschließenden Kapitel 3.

Eine *Custom Activity*, die einen Web Service nutzt, muss daher entweder selbst alle Bestandteile implementieren, die zur Kommunikation mit einem Web Service erforderlich sind. Oder andererseits besteht die Möglichkeit, innerhalb dieser Custom Activity auf ein Web Service Framework zurückzugreifen (vgl. Kapitel 7), das die Nutzung eines Web Services wesentlich vereinfacht. Dieses Vorgehen wurde im Rahmen dieser Arbeit gewählt. Mit Hilfe eines Wizards wird der Vorgang des „Importierens“ von Web Services zudem weiter automatisiert und der Programmierer eines Prozessmodells von Routinearbeiten entlastet, die bei der wiederholten Nutzung von Web Services in einem Prozessmodell entstünden. Der Vorgang des „Importierens“ umfasst

- das Auslesen der durch einen Web Service bereitgestellten Dienste,
- das Erstellen von Proxy-Klassen zur Nutzung der Infrastruktur eines Web Service Frameworks,
- die Erzeugung von vorkonfigurierten *Custom Activities* zur Einbindung in ein Prozessmodell und
- die Darstellung und Verfügbarmachung der erzeugten Aktivitäten im Process Modeler in der *„Activities Gallery“*

Die Aufgabe eines „Web Service Import Wizards“ besteht darin, diese oben genannten Teilaufgaben abzuarbeiten. Diese Teilaufgaben entsprechen den funktionalen Anforderungen. Im Rahmen des praktischen Teils dieser Arbeit wurde ein Prototyp eines solchen „Web Service Import Wizards“ entwickelt. Zum detaillierten Vorgehen bei der Planung und Implementierung siehe Kapitel 8.

2.6 Zusammenfassung

In diesem Kapitel wurde das Umfeld vorgestellt, in dem die Entwicklung der prototypischen Software im Rahmen des praktischen Teils dieser Arbeit stattgefunden hat. Ausgehend von den theoretischen Grundlagen, wie z.B. der Architektur eines Portals und den Zielen der Geschäftsprozessmodellierung mittels Workflow-Management-Prinzipien, wurde die abaXX-Lösung eines Prozessportals beschrieben, die den Ausgangspunkt des praktischen Teils dieser Arbeit darstellt: die Integration und Nutzbarmachung von Web Services im Process Modeler des abaXX-Portals. Dazu wurden Ziele und Anforderungen

formuliert, die die Grundlage für die Implementierung darstellen und in Kapitel 8 detailliert beschrieben werden.

Zunächst aber wird im folgenden Kapitel die Technologie „Web Services“ detailliert vorgestellt, die eine mögliche Umsetzung des übergeordneten Architekturprinzips der sog. Service-oriented Architectures ist.

3 Service-oriented Architectures: Web Services als Instrument zur Anwendungsintegration

In diesem Kapitel werden Konzepte und Ideen einer sog. „Service-Oriented Architecture“ (SOA) vorgestellt. Schwerpunkt der Darstellung ist die Umsetzung einer SOA durch Web Services. Dazu wird zunächst definiert, was Web Services sind, welchen bestehenden Technologien sie ähnlich sind und welche Relevanz ihnen zugeschrieben wird.

3.1 Service-oriented Architectures

Bei einer Service-oriented Architecture (dt. *Service-orientierte Architektur*) handelt es sich um ein Software-Architekturmodell. Eine einheitliche Definition einer SOA existiert nicht, da je nach Schwerpunktsetzung andere Komponenten und Aspekte einer Definition betont bzw. hinzugefügt werden. Im Folgenden wird auf Grund ihrer Generalität eine Definition nach Dostal et al. (2005, 11) gewählt:

> „Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachunabhängige Nutzung und Wiederverwendung ermöglicht.“

Dabei, so die Autoren, weisen SOAs einige grundlegenden Merkmale auf (vgl. Dostal et al. 2005, 9f.):

- **Lose Kopplung:** (engl. *loose coupling*) „Dienste werden von Anwendungen oder anderen Diensten bei Bedarf dynamisch gesucht, gefunden und eingebunden.“ (Dostal et al. 2005, 9) Dies geschieht zur Laufzeit und nicht zur Compile-Zeit der Anwendung. „The notion of loose coupling precludes any knowledge or assumptions about the specific platforms that the requester or the service provider run on, specifics about the implementation that either of the partners uses, or the formats and protocols used to interoperate between them.“ (Weerawarana et al. 2005, 9)
- **Dynamisches Binden und Verzeichnisdienste:** Damit Funktionalitäten dynamisch eingebunden werden können, müssen die gewünschten Dienste gefunden werden. Dies kann mittels eines Verzeichnisdienstes geschehen, der z.B. ähnlich wie die klassischen (Telefonbuch-) Gelben Seiten aufgebaut ist.

- **Verwendung von Standards:** Um Dienste unbekannter Anbieter zu nutzen, müssen die Schnittstellen in maschinenlesbarer, standardisierter Form vorliegen, was beispielsweise durch die Verwendung von offenen Standards erreicht werden kann.
- **Einfachheit und Sicherheit:** Eine breite Akzeptanz einer SOA kann nur erreicht werden, wenn die Nutzung sich relativ einfach gestaltet und Sicherheitsaspekte berücksichtigt werden (siehe hierzu Kapitel 6.1).

Innerhalb einer SOA lassen sich als Teilnehmer die drei Rollen des Dienstanbieters, des Dienstnutzers und des Dienstverzeichnisses beschreiben (siehe Abbildung 3.1), die zur Kommunikation miteinander Nachrichten über Netzwerke, z.B. das Internet, austauschen.

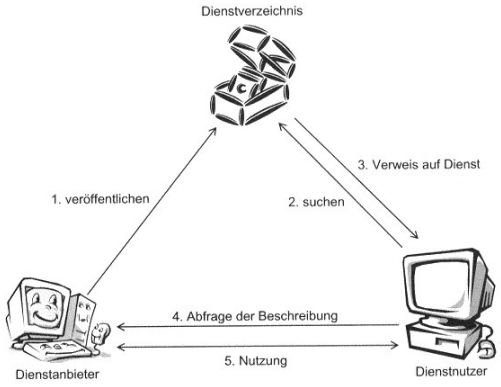

Abbildung 3.1: SOA Dreieck (Dostal et al. 2005, 12)

Ein **Dienst** (engl. *service*) ist ein Programm oder eine Softwarekomponente, deren Schnittstelle in Form einer maschinenlesbaren Dienstbeschreibung vorliegt und dadurch lokal oder remote angesprochen werden kann. Dabei wird das Prinzip der Kapselung realisiert, d.h. es sind nur Zugriffe über die Funktionen der Schnittstelle möglich; die Implementierung wird vor den Nutzern eines Dienstes verborgen (vgl. Dostal et al. 2005, 12). Wenn die Implementierung des Dienstes geändert wird, bleibt im Regelfall die Schnittstellenbeschreibung unverändert erhalten. Greift ein Nutzer einer Anwendung auf den intern geänderten Dienst zu, muss er seine Anwendung nicht modifizieren.

Die **Dienstbeschreibung** (engl. *service description*) beinhaltet eine maschinenlesbare Beschreibung der öffentlichen Schnittstelle des Dienstes, die unabhängig von der verwendeten Implementierung, der Programmiersprache oder Plattform formuliert ist. Meist handelt es sich hierbei um eine funktionale Beschreibung des Dienstes ähnlich der Signatur einer Funktion in einer Programmiersprache, d.h. welche Funktion in welcher Reihenfolge Parameter eines bestimmten Typs erwartet. Eine inhaltliche bzw. semantische Bestimmung eines einzelnen Parameters (z.B. soll der Parameter „String“ eine 5-stellige Zahl beschreiben, die eine deutsche Postleitzahl widerspiegelt) ist mit den heute gängigen Werkzeugen nicht möglich, jedoch im Kontext des Semantic Webs (vgl. Kapitel 9.3) eine zentrale Forderung (vgl. Dostal et al. 2005, 13).

Ein **Dienstanbieter** (engl. *service provider*) „stellt eine Plattform zur Verfügung, welche über ein Netzwerk Zugriff auf mindestens einen Dienst ermöglicht. Damit seine Dienste von Nutzern gefunden werden können, registriert der Dienstanbieter seine Dienste bei einem Verzeichnisdienst." (Dostal et al. 2005, 14) Meist ist der Dienstanbieter ebenfalls für die Gewährleistung des laufenden Betriebs eines Dienstes verantwortlich, wozu z.B. typische Rechenzentrumsaufgaben wie Datensicherung und Wartung zählen (vgl. Dostal et al. 2005, 14).

Der **Dienstnutzer** (engl. *service requestor*) greift über die veröffentlichten Schnittstellen auf die Dienste eines Dienstanbieters zu. Damit beide miteinander kommunizieren können, müssen gemeinsame Protokollstandards verwendet werden.

Das Finden von Diensten wird mittels eines **Dienstverzeichnisses** (engl. *service registry* oder *service directory*) durchgeführt. Dazu muss ein Dienstanbieter seine Dienste selbstständig in einem Verzeichnis registrieren, damit sie von potentiellen Nutzern gefunden werden können[1]. Dostal et al. (2005, 15) gehen davon aus, dass es eine Vielzahl von Dienstverzeichnissen geben wird und dass Firmen sowohl interne (oder private), als auch externe (oder öffentliche) Verzeichnisse oder auch Mischformen (z.B. für die Zusammenarbeit über Unternehmensgrenzen hinweg) für ihre bereitgestellten Dienste unterhalten werden.

Drei Arten von Aktivitäten spielen sich zwischen den Akteuren innerhalb einer SOA ab: Zum Ersten das **Veröffentlichen** eines Dienstes (engl. *publish*), um einen Dienst in ein Dienstverzeichnis einzutragen, nachdem dieser (idealerweise vor dem Veröffentlichen) zur Nutzung bereitgestellt wurde (engl. *deploy*). Zum Zweiten die **Suche** nach Diensten, die beispielsweise basierend auf einer Beschreibung eines Dienstes mögliche Kandidaten liefert, die dann ausgewählt werden müssen. Zum Dritten die erfolgreiche **Bindung** (engl. *binding*) an einen Dienst, um diesen zu nutzen.

3.2 Web Services als Basis einer SOA

Eine Service-oriented Architecture kann beispielsweise mit Web Services (WS) implementiert werden, wofür zum heutigen Zeitpunkt die größte Unterstützung an Werkzeugen und Standards[2] vorhanden ist. Jedoch ist das relativ junge Themengebiet „Web Services" durch eine ständige Weiterentwicklung, Verfeinerung, Festigung und Etablierung von Standards gekennzeichnet. Als Beleg für die zunehmende „Professionalisierung" werten die Autoren Dostal et al. (2005, 26) die Entwicklung der Definition von WS des W3C

[1] Dies wird von Burghardt (2004, 17) mit dynamischer Veröffentlichung bezeichnet. Im Gegensatz dazu steht die statische Veröffentlichung, bei der der Dienstanbieter direkt den Dienstnutzer über die Dienstbeschreibung informiert (vgl. Burghardt 2004, 17).

[2] Auf Grund ihrer weiten Verbreitung bzw. ihres großen Einflusses werden in der IT-Welt häufig auch Spezifikationen als Standards bezeichnet, obwohl es sich eher um De-facto-Standards handelt. Meist werden jedoch viele Spezifikationen nicht oder nur nachträglich von einem offiziellen Standardisierungs-Gremium verabschiedet (vgl. Dostal et al. 2005, 22).

(World Wide Web Consortium[3]), die vom Allgemeinen und Abstrakten (Version Oktober 2002) in eine Definition übergeht, die konkrete Technologien und Spezifikationen benennt (August 2003):

> „A Web Service is a software application identified by a URI, whose interface and bindings are capable of being defined, described, and discovered as XML artifacts. A Web Service supports direct interactions with other software agents using XML-based messages exchanged via internet-based protocols." (Oktober 2002)[4]

> „A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." (August 2003)[5]

Dabei werden die SOA-Komponenten Kommunikation, Dienstbeschreibung und Verzeichnisdienst mit den Web Service Standards SOAP (engl. *Simple Object Access Protocol*), WSDL (engl. *Web Service Description Language*) und UDDI (engl. *Universal Description, Discovery and Integration*) konkretisiert, die in Kapitel 4 näher erläutert werden. Es ergibt sich folgende „Dreiecksbeziehung" (siehe Abbildung 3.2):

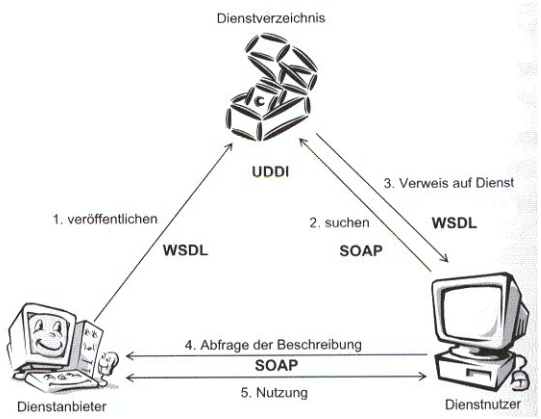

Abbildung 3.2: Web Services Architektur-Dreieck (Dostal et al. 2005, 28)

Ein Dienstanbieter erstellt mittels eines XML-basierten[6] WSDL-Dokuments eine Schnittstellenbeschreibung seines Dienstes, den er bereitstellen möchte. Dieses WSDL-Dokument

[3] `http://www.w3c.org`, verifiziert am 13.02.2006, 18:45 MEZ
[4] `http://www.w3.org/TR/2002/WD-wsa-reqs-20021011`, verifiziert am 19.01.2006, 16:09 MEZ
[5] `http://www.w3.org/TR/2003/WD-ws-gloss-20030808/`, verifiziert am 19.01.2006, 16:14 MEZ
[6] eXtensible Markup Language,

kann in einem UDDI-basierten Verzeichnisdienst veröffentlicht werden, den ein Dienstnutzer auf der Suche nach Diensten konsultieren kann. Der Dienstnutzer erhält als Suchergebnis eine URI-Referenz auf die WSDL-Datei des gewünschten Web Service. Diese WSDL-Beschreibung kann er zum Einbinden des gewünschten Web Service innerhalb seiner Anwendung nutzen, um letztlich z.B. über das Protokoll SOAP mit dem Dienstanbieter zu kommunizieren (vgl. Dostal et al. 2005, 29).

3.3 Einordnung von Web Services in verwandte Technologien

Eine SOA kann nicht nur mit Web Services umgesetzt werden. Für die Entwicklung von verteilten Anwendungen existieren bereits seit Längerem Technologien, wie z.B. die Common Object Request Broker Architecture (CORBA), das von Sun Microsystems entwickelte Konzept der JAVA RMI (Remote Method Invocation) und der Ansatz des Distributed Common Object Model (DCOM) von Microsoft. Da, so Dostal et al. (2005, 38), die Ansätze von Sun und Microsoft plattformspezifisch sind, wird im Folgenden auf die plattformunabhängige Lösung CORBA näher eingegangen.

Innerhalb der CORBA-Spezifikation bildet der Object Request Broker (ORB) das zentrale Vermittlungselement zwischen den beteiligten Objekten. Er übernimmt die gesamte Abwicklung der Kommunikation. Zur Kommunikation mit dem ORB wird ein Application Programming Interface (API) eingesetzt, das für mehrere Programmiersprachen vorhanden ist. „Der ORB kapselt gegenüber dem Objekt die Spezifika der genutzten Hardware, des Betriebssystems sowie des Netzwerkprotokolls. Dabei werden die Objekte durch eine Schnittstelle, die durch die Interface Definition Language (IDL) beschrieben wird, vom ORB separiert. Die Schnittstellenbeschreibung in Form einer IDL lässt sich unter Anwendung eines IDL-Compilers in jede beliebige Programmiersprache überführen. Die Kommunikation der ORBs untereinander sowie der Objekte mit dem ORB erfolgt über das Internet Inter-ORB Protocol (IIOP) und somit über ein speziell standardisiertes Protokoll. Die Übertragung der Daten zwischen den Objekten erfolgt in binärer Form (Common Data Representation, CDR).“ (Burghardt 2004, 23f.) Dostal et al. (2005, 38) fassen die Unterschiede zwischen CORBA und Web Services zusammen, siehe Tabelle 3.1.

	CORBA	**Web Services**
Protokoll	IIOP u.a.	SOAP, HTTP, XML Schema
Endpunktreferenzierung	URL u.a.	URL
Schnittstellenbeschreibung	IDL	WSDL
Naming, Verzeichnis	Naming Service, Interface Repository, Trader Service	UDDI
Nutzdaten	binäre	XML

Tabelle 3.1: CORBA im Vergleich zu Web Services (Dostal et al. 2005, 38)

3.4 Einsatzmöglichkeiten von Web Services

Betrachtet man die Marketing-Verlautbarungen von Herstellern, Beratern oder anderen im Web Service Bereich involvierten Personenkreisen, könnte man beinahe zum Schluss gelangen, mit Web Services wäre der „Heilige Gral“ der Informationstechnologie gefunden, mit dem sich sämtliche Probleme in der IT-Welt lösen lassen. Die (erhofften) Vorteile werden gerne aufgezählt, jedoch müssen diese (Wunsch-)Ziele einer Überprüfung in der Realität mit den dort zu Verfügung stehenden Mitteln erst Stand halten.

Ein häufig genannter Vorteil bestünde darin, dass durch die Nutzung von Web Services keine Programmierarbeit mehr anfiele. „Solange mit Hilfe von Web Services nur einfache Beispielanwendungen, wie Temperaturumrechnung und Aktienkursabfragen, implementiert werden sollten, war diese Annahme sicher richtig.“ Problematisch und ohne zusätzlichen Programmieraufwand heute nicht umzusetzen sind z.B. das Verwenden von anwendungsspezifischen Datentypen sowie das gleichzeitige und koordinierte Ansprechen von mehreren Web Services innerhalb einer Anwendung (vgl. Dostal et al. 2005, 39f.).

Außerdem kann der fälschliche Eindruck entstehen, dass Web Services *per definitionem* interoperabel seien. Web Services nutzen zwar XML-Dokumente zum Datenaustausch, jedoch beseitigt dieser Umstand allein keine Interoperabilitätsprobleme. Die Hersteller von Web Service Komponenten müssen daher versuchen, obwohl eine Spezifikation in der Regel einen gewissen Spielraum zur Interpretation bereitstellt (z.B. zur Wahrung der Abwärtskompatibilität mit Vorgängerversionen), ihre jeweiligen Produkte interoperabel zu gestalten – bspw. durch regelmäßigen Austausch in einem Gremium wie der WS-I (Web Service Interoperability Organization[7]) (vgl. Dostal et al. 2005, 40f.).

Die WS-I verabschiedet keine Standards, sondern „[...] betrachtet konkrete Spezifikationen und deren Implementierungen verschiedener Hersteller zu Web-Services [...]“. In sog. Profiles wird beschrieben, „[...] wie die Implementierungen der unterschiedlichen Hersteller zu nutzen sind, damit die Interoperabilität erhalten bleibt.“ (Dostal et al. 2005, 37). In den folgenden Abschnitten werden zunächst die Vorteile aus technischer, anschließend aus betriebswirtschaftlicher Sicht aufgezeigt, um Einsatzmöglichkeiten für Web Services aufzuzeigen.

3.4.1 Vorteile aus technischer Sicht

Ort (2004) benennt folgende Vorteile von Web Services aus technischer Sicht:

- **Wiederverwendbarkeit:** Entwickler können auf bereits vorhandenen Programmcode zurückgreifen und diesen über Web Services innerhalb und auch zwischen Unternehmen nutzbar machen. Dadurch werden Doppelentwicklungen vermieden, was

[7] `http://www.ws-i.org`, verifiziert am 09.01.2006, 17:12 MEZ

letztlich Zeit und Kosten spart. Zudem können Web Services bei der Anwendungsintegration eingesetzt werden, um so z.B. Alt-Systeme (engl. *legacy systems*) relativ einfach mit anderen Systemen zu verbinden.

- **Interoperabilität:** „Web Services comprise a maturing set of protocols and technologies that are widely accepted and used, and that are platform, system and language independent. In addition, the protocols and technologies work across firewalls, making it easier for business partners to share vital services."
- **Skalierbarkeit:** „Because services in an SOA are loosely coupled, applications that use these services tend to scale easily – certainly more easily than applications in a more tightly-coupled environment."
- **Flexibilität:** „Loosely-coupled services are typically more flexible than more tightly-coupled applications. In a tightly-coupled architecture, the different components of an application are tightly bound to each other, sharing semantics, libraries, and often sharing state."
- **Kosteneffizienz:** Werden Alt-Systeme oder Anwendungen von Geschäftspartnern integriert, geschieht dies oft durch maßgeschneiderte Lösungen, die in der Planung, Erstellung und Wartung teuer sind. Web Services dagegen bieten mit ihren auf offenen Standards basierenden Komponenten eine kostengünstigere Integrationsmöglichkeit, da auf Grund der losen Kopplung beispielsweise keine starke Verzahnung der zu integrierenden Komponenten bei der Planung zu berücksichtigen ist. Dies wirkt sich auch auf den späteren Betrieb und die Wartung aus.

3.4.2 Vorteile und Anwendungsszenarien aus betriebswirtschaftlicher Sicht

Das Themengebiet „Web Services" kann entweder aus einer technischen Perspektive (die in dieser Arbeit gewählt wurde) oder aber aus einer betriebswirtschaftlichen, prozessorientierten Perspektive betrachtet werden. Hierbei stehen Themen wie Prozessintegration, Leistungsbeschreibung oder Geschäftsmodelle im Vordergrund.

Zur Verdeutlichung der zukünftigen Nutzung und Potentiale von Web Service-Anbietern zitiert Reichmayr (2003, 97) eine im Jahre 2001 von Schatsky[8] durchgeführte Studie – wobei man heute darauf hinweisen muss, dass diese Studie wohl zu einem Zeitpunkt entstand, zu dem das Thema „Web Services" in der Berichterstattung einen großen Hype erfahren hatte. Dennoch sollen diese Ergebnisse hier präsentiert werden, um die damalige Tendenz und die Erwartungen aufzeigen. Er ermittelte, dass:

- 16% der Unternehmen Web Services im kommenden Jahr (d.h. 2002) einsetzen, um neue Geschäftspartner zu finden und mit diesen zusammen zu arbeiten,
- 60% der Unternehmen Web Services für interne Applikationen nutzen werden,

[8] Schatsky, D.: Web Services: Invest Today for Cost Savings, Not Flashy Business Models. Jupiter Media Metrix, New York. 2001

- 53% der Unternehmen Web Services zur Interaktion mit bestehenden Lieferanten und Partnern einsetzen werden und
- 23% der Unternehmen im kommenden Jahr keine Web Services einsetzen werden.

Reichmayr verwendet hierbei den Begriff des *Out-taskings*, der eine Spezialform des Outsourcings ist. Während bei einem klassischen Out-sourcing-Prozess in der Regel vorwiegend Unterstützungsprozesse ausgelagert werden, werden im Rahmen des Out-taskings einzelne Aufgaben aus den Leistungsprozessen ausgegliedert. Dies sind beispielsweise die Auftrags- oder Paketverfolgung, Lagerbestandsabfragen oder Kreditkartentransaktionen. Diese ausgegliederten Prozesse können mit Web Services integriert werden (vgl. Reichmayr 2003, 99f.). Silberberger (2003, 84ff.) benennt – zum heutigen Zeitpunkt auf Grund ihrer unscharfen Formulierung euphorisch anmutende – Anwendungsszenarien für Web Services, die im Folgenden ausschnittsweise wiedergegeben werden:

- **Öffentlicher Sektor:** „Behörden beschäftigen sich mit riesigen Datenmengen. Doch meist sind die zugehörigen Datenspeicher nur wenigen, voneinander isolierten Nutzergruppen zugänglich. Dabei könnten in vielen Fällen aus der Zusammenarbeit unterschiedlicher Organisationseinheiten und Institutionen erhebliche Synergieeffekte resultieren, könnten Doppelarbeiten vermieden und arbeitsintensive Medienbrüche in der Zusammenarbeit minimiert werden. Je nach Natur der Daten können Web Services hier enormen Mehrwert stiften.“
- **Versicherungen:** Die Versicherungswirtschaft verfügt über umfangreiche Kunden-, Markt- und Schadensdaten. Sie sind meist zentral gespeichert, werden jedoch überwiegend durch ein dezentrales Netz von zum Teil unabhängigen Vertriebspartnern (z.B. Maklern) eingegeben. „Da im Schadensfall weitere externe Kooperationspartner wie zum Beispiel Kfz-Gutachter, Ärzte oder Bausachverständige in einen idealerweise zügig abzuwickelnden Regulierungsprozess eingebunden werden müssen, bieten sich zahlreiche Schnittstellen- und Integrationsaufgaben, die mit Hilfe von Web Services gelöst werden können.“
- **Reiseindustrie:** „Für Geschäftsreisende können via Web Services die Angebote verschiedener Anbieter zum Beispiel Fluggesellschaften, Hotelketten, Autovermietungen und Bahngesellschaften in einer Weise zu Angebotspaketen aggregiert und kombiniert werden, die automatisch die sich permanent ändernde Nutzerhistorie sowie aktiv eingegebene Präferenzen berücksichtigt.“

3.5 Enterprise Service Bus

Wie bereits in der Einleitung zu Service-oriented Architectures motiviert wurde, kann eine SOA ein geeignetes Werkzeug zur Anwendungsintegration sein. Ziel der sog. „Enterprise Application Integration“ (EAI) ist es, bislang autonome Anwendungen und Systeme miteinander zu koppeln, um einen gemeinsamen Datenaustausch über Plattformgrenzen

hinweg (z.B. Hardware, Betriebssysteme) zu ermöglichen. Die Notwendigkeit zur Integration von Anwendungen ist in den seltensten Fällen ein im Voraus geplantes Vorhaben. Meist ergibt sich ein Impuls zur Integration durch äußere Umstände, wie z.B. durch eine Fusion von zwei Unternehmen und dem gewünschten gemeinsamen Zugriff auf Anwendungen und Daten.

Um die Integration von Anwendungen zu ermöglichen, bieten sich zum einen individuell erstellte Lösungen an, um auf diese Weise eine Anwendung jeweils mit einer anderen Anwendung zu koppeln. Jedoch weist dieses Punkt-zu-Punkt-Vorgehen gravierende Nachteile auf: Werden zukünftig weitergehende Integrationsmaßnahmen erforderlich, so muss ein evtl. entstandenes, sehr komplexes System angepasst werden. Das kann schnell sehr aufwändig und kostenintensiv werden.

Einen anderen Ansatz zur Integration stellt der Einsatz von EAI-Produkten dar. Klassische EAI-Software arbeitet nach dem *Hub-and-Spoke*-Prinzip (vgl. Lorenzelli-Scholz 2005, 18). Über eine zentrale Komponente, den EAI-Hub (z.B. realisiert durch eine „Message oriented Middleware“ [MOM]), werden Daten und Mitteilungen vermittelt. „Dazu verwendet der EAI-*Hub* spezielle Adapter, die die eingehenden Daten in ein kanonisches Format transformieren, das wiederum vom *Hub* und vom Adapter verstanden wird. Diese Adapter stellen zuweilen recht komplexe, mehrschichtige Softwaresysteme dar, die sowohl die Interaktion mit dem EAI-*Hub* als auch zwischen den Endpunkten der Applikationen sicherstellen müssen.“ (Lorenzelli-Scholz 2005, 18) Als nachteilig an diesem Vorgehen erweisen sich die oft hohe Investition in eine EAI-Softwarelösung sowie die eher erzwungene Homogenisierung der Middleware- und EAI-Produkt-Komponenten, die das Risiko der Herstellerabhängigkeit birgt. Außerdem, so Lorenzelli-Scholz (2005, 18), stoßen viele Implementierungen schnell an ihre Grenzen, wenn zusätzlich ein Transaktions- oder Sicherheitskontext übermittelt werden muss.

Das Konzept des *Enterprise Service Bus* (ESB) eröffnet zusammen mit einer SOA eine weitere Möglichkeit zur Anwendungsintegration. Ein ESB steht nicht im Gegensatz zu einer SOA, sondern beide ergänzen sich. Während eine SOA Regeln und Muster repräsentiert, um Applikationen als Services nutzbar zu machen (z.B. durch WSDL), übernimmt ein ESB konkrete Integrations- und Überbrückungsfunktionen (vgl. Abbildung 3.3). Im Moment existiert keine einheitliche, umfassende Definition eines ESBs, weshalb im Folgenden zur Beschreibung grundlegende Funktionen eines ESBs aufgezeigt werden (vgl. hierzu Dostal et al. (2005, 20) und Schmidt et al. (2005, 781ff.)):

- Ein ESB beinhaltet keine Geschäftslogik und dient ausschließlich als Infrastrukturkomponente für den Datenfluss. Nachdem ein ESB die Nutzdaten zu einem Endpunkt transportiert hat, endet seine Zuständigkeit. Den Dienstnachfragern oder Dienstanbietern erscheint der ESB transparent, d.h. für sie macht es keinen Unterschied, ob sie direkt oder über den ESB angesprochen werden.
- **Protokollunabhängigkeit:** Ein gemeinsames Kommunikationsprotokoll ist in einem ESB nicht vorgeschrieben. Schmidt et al. (2005, 783) gehen davon aus, dass wahr-

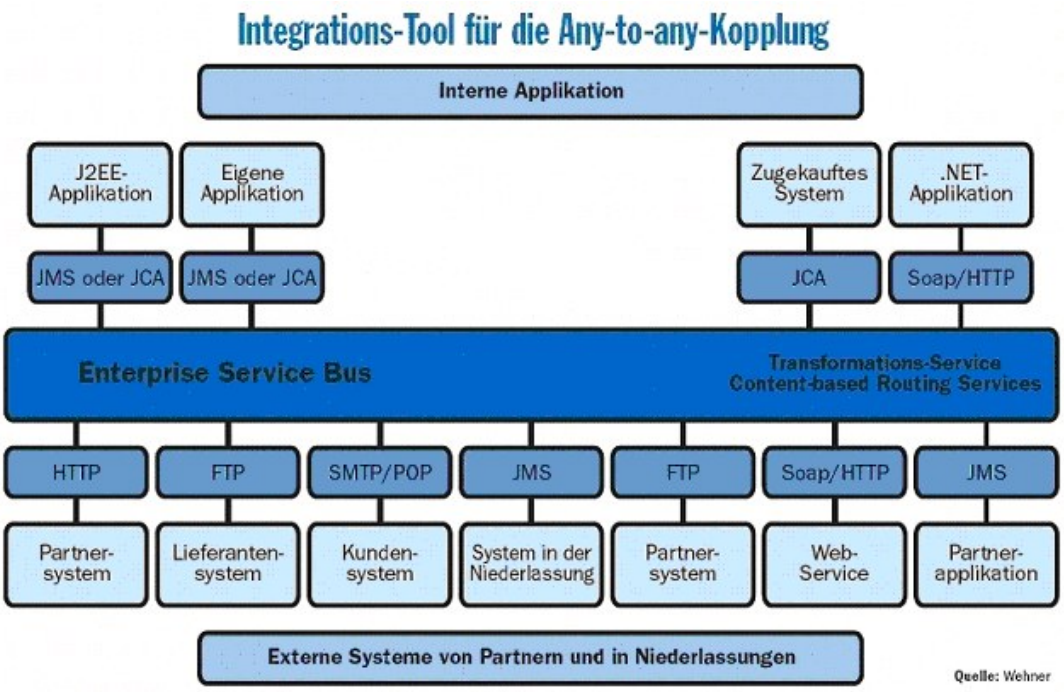

Abbildung 3.3: Architekturvariante eines Enterprise Service Bus (Wehner 2005)

scheinlich die meisten Container, die die Implementierung von Services beinhalten, über SOAP/HTTP-Protokolle angesprochen werden.

- **Dynamische Anpassung:** Dienste werden in der ESB-Registry verwaltet. Dort sind Metadaten hinterlegt, die einen Dienst beschreiben, z.B. die Service-Endpunkte oder Beschreibungen der Dienstqualität (Quality of Service, QoS). Zum Beschreiben der Service-Endpunkte und Schnittstellen kann beispielsweise WSDL eingesetzt werden. Sollen weitere Bedingungen oder Einschränkungen (also alles, was nicht mit der Schnittstellendefinition zu tun hat) formuliert werden, sind *policy declarations* erforderlich. Sie können zur Beschreibung des Verhaltens oder der Semantik eines Dienstes genutzt werden und z.B. mit dem Standard WS-Policy (vgl. Kapitel 6.1.2) ausgedrückt werden. „All the meta-data, conditions, and constraints used to enable a connection from a requestor to a provider can be discovered, used, and modified at runtime." (Schmidt et al. 2005, 783) Änderungen erfordern somit keine Anpassung und Änderung auf Seiten der Dienstnutzer bzw. Dienstanbieter, sondern sie erfolgen dynamisch.
- **Mediation:** Zum Erreichen dieser Flexibilität bezüglich der Protokollunabhängigkeit und der dynamischen Anpassung müssen Nachrichten z.B. zwischen SOAP/HTTP nach SOAP/JMS (Java Messaging Service) transformiert werden oder automatisch ein anderer Dienstanbieter gesucht werden, falls die Antwortzeit eines Services unterhalb eines zuvor definierten Schwellenwerts sinkt. Schmidt et al. (2005, 784) bezeichnen diese Eigenschaften eines ESB mit *Mediation* und formulieren in ihrem Artikel mehrere *mediation patterns*, die miteinander kombiniert werden können und somit als Bausteine eines ESBs gelten. „Through these and other systems manage-

ment capabilities, the ESB ensures that a loosely coupled and dynamically varying SOA is still manageable in a production environment.“ (Schmidt et al. 2005, 784):

- *monitor pattern* zum Beobachten von Nachrichten;
- *transcoder pattern* zum Ändern des Nachrichtenformats z.B. von SOAP in eine JMS-Nachricht;
- *modifier pattern* zum Aktualisieren von Nachrichten durch Transformation (z.B. Ver- oder Entschlüsseln von Nachrichten für den Sender/Empfänger) oder „Enrichment“ (z.B. Hinzufügen von Informationen aus externen Datenquellen);
- *validator pattern* zum Überprüfen, ob eine Nachricht dem Empfänger zugestellt werden darf;
- *router pattern* zum Ändern der Route einer Nachricht z.B. auf Grund von Lastverteilungsaspekten;
- *discovery pattern* zum dynamischen Auswählen von Dienstanbietern auf Basis der vom Dienstnachfrager formulierten Bedingungen;
- *clone pattern* zum Kopieren einer Nachricht und Abändern seiner Route;
- *aggregator pattern* zum Sammeln und zusammengefassten Versenden von Nachrichten;

Vergleicht man die Funktionen eines ESBs mit denen von CORBA stellt man fest, dass die Mediator-Fähigkeiten eines ESBs zum Teil durch bereits seit längerem bestehende Funktionen eines ORBs abgedeckt werden könnten. Beide Technologien, ESB und CORBA, weisen insgesamt betrachtet ein ähnliches Komplexitätsprofil auf. Auf Grund der Verwendung von verbreiteten Standards, wie z.B. XML, erfährt die ESB-Technologie zurzeit eine umfangreichere Werkzeug-Unterstützung, als es bei CORBA der Fall ist (vgl. hierzu Trenaman 2005).

Der ESB ist ein Architekturkonzept, das die Vorteile einer SOA nutzt und gleichzeitig zu ihrem erfolgreichen Einsatz beitragen kann. „Man könnte sagen, SOA ist ein Auto und ESB eine Straße, auf der das Auto fährt.“ (Wehner 2005)

3.6 Zusammenfassung

In diesem Kapitel wurden Service-oriented Architectures als Architekturkonzept vorgestellt. In einer möglichen Umsetzung durch Web Services eröffnen sich mit dieser Technologie vielfältige Einsatzmöglichkeiten. Auf Grund der Nutzung von breit akzeptierten Standards kann eine große Interoperabilität zwischen verschiedenen Herstellern ermöglicht werden. Beispielsweise lassen sich mit Web Services Legacy-Systeme einfacher miteinander koppeln oder verschiedene betriebliche Anwendungen integrieren (EAI), was Raum für weitere flexible Nutzungsszenarien in der Zukunft schafft. In diesem Umfeld ist besonders die weitere Entwicklung des ESB-Konzepts zu berücksichtigen, das die Vorteile einer SOA nutzt, um Informationsflüsse auf einer weiteren Abstraktionsebene,

z.B. innerhalb eines Betriebs, zu modellieren und dabei ein Höchstmaß an Flexibilität zu gewährleisten.

Im nächsten Kapitel werden die Architektur und einzelne Komponenten von Web Services vorgestellt sowie einzelne Protokolle und Standards erläutert, wobei als Ausgangspunkt das Konzept einer Service-oriented Architecture herangezogen wird. Dieses folgende Kapitel soll die Grundlagen für das Verständnis der eingesetzten Technologie „Web Services“ schaffen und einen Überblick über die technologische Umsetzung von Web Services bieten.

4 Architektur und Komponenten von Web Services

Um eine Service-oriented Architecture umzusetzen, werden verschiedene Technologien und Spezifikationen eingesetzt. Im Falle ihrer Umsetzung durch Web Services erhält man einen *Web Service Stack*, wobei eine Variante dieses Stacks in Abbildung 4.1 dargestellt wird. Jede Schicht des Stapels stellt der darüber liegenden Schicht Funktionalitäten zur Verfügung, die für die Umsetzung der Gesamtarchitektur als Bausteine erforderlich sind.

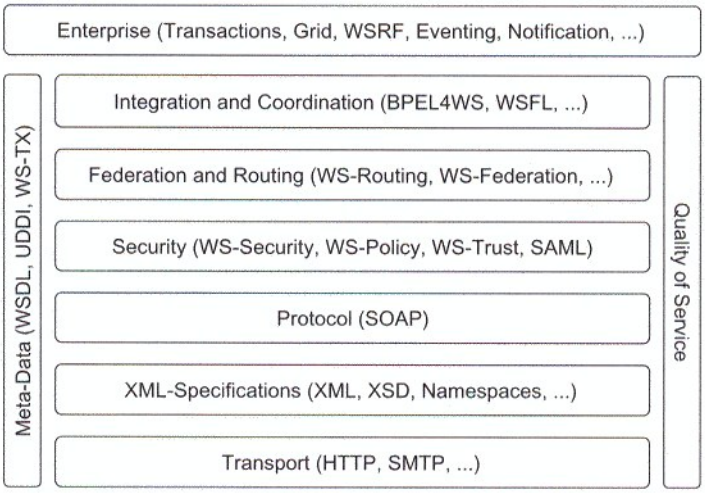

Abbildung 4.1: Web Services Stack (Dostal et al. 2005, 29)

XML als textbasierte Meta-Auszeichnungssprache dient dabei der Beschreibung, dem Austausch und der Manipulation von strukturierten Daten, was eine plattformübergreifende Datennutzung ermöglicht (vgl. Burghardt 2004, 41). Die XML-Nachrichten werden auf der Transportschicht vom Sender zum Empfänger übertragen. Dies geschieht bei Web Services zum jetzigen Zeitpunkt überwiegend mittels HTTP; es muss aber nicht zwangsläufig so gestaltet werden. Zum Austausch der Nachrichten wird die SOAP-Spezifikation eingesetzt, die selbst auf XML basiert. Auf der nächst höheren Ebene können XML-spezifische Sicherheitskonzepte eingeführt werden, die z.B. für eine Verschlüsselung bzw. Authentifizierung und Autorisierung sorgen. „Die fünfte Schicht (Federation and Routing) verfeinert die bisher genannten Konzepte und regelt das Zusammenspiel unterschiedlicher Spezifikationen untereinander im Kontext jeweils eines Web Services [...]“ (Dostal et al. 2005, 30) Soll eine Aufgabe durch das Zusammenspiel von mehreren Web Services bearbeitet werden, kommen die Spezifikationen der Schicht *Integration und Coordination* zum Einsatz. An den vertikalen Seiten befinden sich – links dargestellt – Meta-Daten-Komponenten, die einen Web Service beschreiben oder auffindbar machen,

sowie – rechts – eine Quality of Service Komponente, um die Güte eines angebotenen Dienstes zu beschreiben (vgl. Dostal et al. 2005, 30).

Das Themengebiet „Web Services“ ist ein noch relativ junges Themengebiet. Zu einem konkreten Aspekt können daher unterschiedliche Ansätze oder Spezifikationen vorhanden sein. Dostal et al. (2005, 30) gehen davon aus, dass ca. 100 verschiedene Standards zum Thema Web Services in Gebrauch oder noch in der Entstehung sind, die sich sogar teilweise überdecken. Außerdem hängen oftmals Spezifikationen voneinander ab, so dass die Komplexität, v.a. für Neueinsteiger in dieses Themengebiet, stark zunimmt (beispielsweise benutzt das Protokoll SOAP die W3C-Empfehlung zu *„XML Schema“*, um die Struktur eines XML-Dokuments zu beschreiben).

4.1 Kommunikation mittels SOAP

Zur Kommunikation untereinander senden und empfangen Web Services SOAP-Nachrichten. Die SOAP-Spezifikation legt fest, wie eine Nachricht formatiert sein muss, um als SOAP-Nachricht zu gelten und dementsprechend weiter verarbeitet zu werden. Den Ausführungen in diesem Kapitel liegt die SOAP-Version 1.2 zu Grunde. Nach einer generellen Beschreibung des Aufbaus einer SOAP-Nachricht wird auf wichtige, später auch für den praktischen Teil der Arbeit relevante Besonderheiten eingegangen.

4.1.1 Aufbau einer SOAP-Nachricht

Die SOAP-Spezifikation in der Version 1.1 entstand durch ein Firmenkonsortium im Mai 2000 und begründete das Akronym des *Simple Object Access Protocols*. Jedoch wurde mit der aktuellen Version 1.2, die durch das W3C gepflegt und weiter entwickelt wird, auf eine Bedeutung als Akronym des Begriffs „SOAP“ verzichtet. Daher ist SOAP von nun an als Protokollbezeichnung zu verstehen[1].

SOAP „ist ein XML-basiertes und durch die Nutzung der Metasprache XML ein einfaches, sprach- und plattformunabhängiges Kommunikationsprotokoll zum Austausch strukturierter Informationen.“ (Burghardt 2004, 50) Eine einheitliche Strukturierung ist deswegen nötig, damit die Kommunikationspartner ermitteln können, welche Teile einer Nachricht die Nutz- oder Steuerungsdaten beinhalten. Eine SOAP-Nachricht besteht aus bis zu drei Teilen (siehe Abbildung 4.2), dem *SOAP-Envelope* als Root-Element der SOAP-Nachricht, einem optionalen *SOAP-Header* und dem *SOAP-Body*, der die eigentlichen Nutzdaten (engl. *payload*) enthält. Der SOAP-Header ermöglicht es, durch Hinzufügen

[1] Die SOAP Spezifikation der Version 1.2. ist in mehrere Dokumente aufgeteilt: Part 0 (Primer) als eine Art Tutorial ohne normativen Charakter (`http://www.w3.org/TR/2003/REC-soap12-part0-20030624/`), Part 1 (Messaging Framework) beschreibt den Aufbau einer SOAP-Nachricht (`http://www.w3.org/TR/2003/REC-soap12-part1-20030624/`), Part 2 (Adjuncts) befasst sich u.a. mit der Codierung von Daten via SOAP (`http://www.w3.org/TR/2003/REC-soap12-part2-20030624/`) (alle URLs verifiziert am 09.01.2005, 17:13 MEZ).

von „header blocks“ als Kind-Elemente Funktionalitäten in SOAP modular zu ergänzen. „SOAP defines several well-known attributes that you can use to indicate who should deal with a header block and whether processing of it is optional or mandatory.“ (Weerawarana et al. 2005, 37).

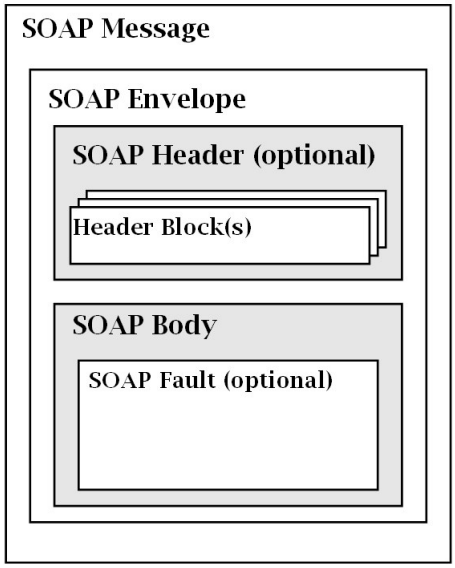

Abbildung 4.2: SOAP Nachricht

Eine Nachricht kann entweder direkt vom Sender zum Empfänger übermittelt werden oder über mehrere Zwischenstationen (engl. *intermediaries*) übertragen werden; jede Station in der Übertragungskette (engl. *message path*) ist ein Knoten (engl. *node*). Wenn eine Zwischenstation auf einen für sie bestimmten Header-Block innerhalb einer SOAP-Nachricht stößt, werden die dafür definierten Aktionen durchgeführt (z.B. wird geprüft, ob eine Autorisierung vorliegt). Ebenfalls können Nachrichten sowohl auf Basis ihres Headers, als auch ihrer Payload geroutet werden: „Note that such benefits were not available in architectures such as DCOM, CORBA and Java Remote Method Invocation (RMI), where protocol headers were infrastructural details opaque to the application.“ (Weerawarana et al. 2005, 38)

Kann eine Nachricht von einem Kommunikationspartner nicht verarbeitet werden, wird ein SOAP-Fehler (engl. *SOAP fault*) als Antwortnachricht versandt. Dabei stellt der Fault Block das einzige Kind-Element des SOAP-Bodys dar, das aus zwei Pflichtelementen und weiteren optionalen Elementen besteht. Das Pflichtelement *„Code“* gibt mit einer in der SOAP-Spezifikation festgelegten Codierung die Fehlerquelle an und im zweiten Pflichtelement *„Reason“* wird eine für den Menschen lesbare Fehlerbeschreibung zurück geliefert.

Die SOAP-Spezifikation schreibt nicht vor, mit welchem Transportprotokoll eine Nachricht übertragen werden muss. „Die Wahl des Transportprotokolls hängt von den jeweiligen Anforderungen ab. Wenn ein Protokoll benötigt wird, das praktisch überall zur Verfügung steht, so ist HTTP sicherlich eine gute Wahl. Wird allerdings ein gewisses

Maß an Übertragungssicherheit (ähnlich einem Einschreiben) benötigt, dann sollte eher ein Messaging-System wie WebSphereMQ gewählt werden. Aufgrund seiner Herkunft aus dem Internet-Umfeld ist das zurzeit am häufigsten genutzte Transportprotokoll für SOAP-Nachrichten natürlich HTTP.“ (Dostal et al. 2005, 61f.)

4.1.2 Interaction styles und data encoding styles

Zwei Faktoren beeinflussen das Aussehen einer SOAP-Nachricht: Die Form der Interaktion (engl. *interaction style*) und die Art und Weise, wie die Nutzdaten in XML codiert werden (engl. *encoding rules* oder *encoding style*). Sie werden im Rahmen der Dienstbeschreibung innerhalb eines WSDL-Dokuments definiert und wirken sich auf die konkrete Gestaltung einer SOAP-Nachricht aus (vgl. Kapitel 4.2.2.3).

4.1.2.1 Interaction styles

Die Interaktion kann entweder im *document style*, bei dem sich die Kommunikationspartner auf die Struktur der ausgetauschten Dokumente einigen oder im *RPC style* (Remote Procedure Call) abgewickelt werden (siehe Abbildung 4.3). Im zweiten Fall beinhaltet der Body des SOAP-Request statt eines Dokuments den eigentlichen Methodenaufruf und die Methodenparameter (im linken Teil der Abbildung 4.3 (b) die Methode „orderGoods“) (vgl. Alonso et al. 2004, 158 f.).

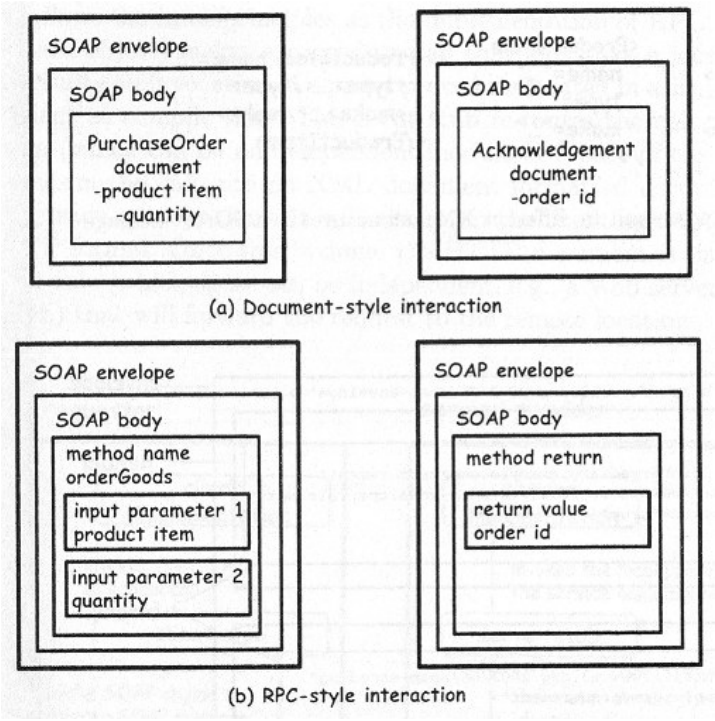

Abbildung 4.3: Interaktions-Typen bei SOAP (Alonso et al. 2004, 159)

4.1.2.2 Data encoding styles

Um Nutzdaten innerhalb einer SOAP-Nachricht zu übertragen, müssen diese vom Sender in ein Format gebracht werden, dass der Empfänger versteht. Dazu müssen zunächst die Objekte zunächst serialisiert werden (engl. *(to) serialize)*, d.h. sie werden in Bytes konvertiert, die über eine Netzwerkverbindung übertragen werden können. Soll sichergestellt werden, dass ganze Objekte incl. ihres Datentyps übertragen werden, muss ein weiterer Mechanismus genutzt werden, der auf den zuvor serialisierten Daten aufbaut. Dieser Mechanismus wird mit (engl.) *marshalling* bezeichnet. Auf Empfängerseite muss die codierte Nachricht wieder entpackt werden („deserialisieren“ oder engl. *(to) deserialize*, bzw. bei Übermittlung von Datentypen [engl.] *unmarshalling*), um die Daten zu nutzen.

Zuvor müssen sich die Kommunikationspartner über das „Wie“ des Codierens einigen, was bei SOAP mit *encoding style* bezeichnet wird: „This defines how data structures including basic types such as integers and strings as well as complex types such as arrays and structures can be serialized into XML.“ (Alonso et al. 2004, 159) Die SOAP-Spezifikation schreibt keine Codierungsverfahren zwingend vor. Neben dem mit SOAP eingeführten *SOAP encoding* ist u.a. auch eine auf XML Schema[2] basierende Codierung möglich, die als *literal encoding* bezeichnet wird. Soll z.B. ein Objekt des Typs „ProductItem“ übertragen werden, kann man je nach gewählter Codierungsart folgende beispielhafte XML-Repräsentation erhalten (aus Alonso et al. 2004, 160):

```
<ProductItem>
<name>...</name>
<type>...</type>
<make>...</make>
</ProductItem>
```

```
<ProductItem
name="..."
type="..."
make="..."
/>
```

```
<ProductItem name=„...“
<type>...</type>
<make>...</make>
</ProductItem>
```

4.1.3 WS-Addressing

Das Binden einer SOAP-Nachricht an ein Transportprotokoll (engl. *binding*) bestimmt z.B. das Format, in dem eine Nachricht für den Transport mit einem bestimmten Protokoll aufbereitet werden muss. Beispielsweise kann eine SOAP-Nachricht innerhalb eines HTTP-Requests verschickt werden, wobei entweder die Operationen GET oder POST eingesetzt werden können. Genauso gut ist der Versand als E-Mail mittels SMTP (Simple Mail Transfer Protocol) möglich.

Die Bindung an ein Transportprotokoll hat zugleich die Funktion der Adressierung (engl. *addressing*), um einen Service-Endpunkt (engl. *service endpoint*) als Ziel eines Web Service Aufrufes anzusprechen. Die Adresse des Service-Endpunkts ist nicht Bestandteil der

[2] Status des Dokuments: W3C Recommendation, `http://www.w3.org/TR/xmlschema-0/`, verifiziert am 10.01.2006, 16:47 MEZ

SOAP-Nachricht. „This is resolved by including the SOAP message as part of an HTTP request or as part of an SMTP message“. (Alonso et al. 2004, 162) Dadurch wird keine Unabhängigkeit des SOAP-Funktionsumfangs vom jeweils darunterliegenden Transportprotokoll erreicht. Außerdem, so Dostal et al. (2005, 65), „[...] verhindert diese enge Kopplung eine abstrakte und neutrale Formulierung von Service-Endpunkten auf der Ebene von SOAP [...], da die Menge der relevanten Informationen für den Endpunkt abhängig vom jeweiligen Transportprotokoll ist.“ Alonso et al. (2004, 191) verdeutlichen dies mit dem Beispiel, dass innerhalb einer SMTP-Nachricht das „to“-Feld mittels einer URI (Uniform Resource Identifier) den Endpunkt eines Dienstes beschreibt, es jedoch keine Möglichkeit gibt, eine Standard-SMTP-Nachricht mit einem zusätzlichen Info-Feld z.B. zur Verwaltung einer Kunden-ID hinzuzufügen.

Diese Unzulänglichkeiten der SOAP-Spezifikation sollen durch Entwicklung des Standards *WS-Addressing* ausgeglichen werden, an dessen Erarbeitung mehrere namhafte Unternehmen beteiligt sind. Die Spezifikation wurde inzwischen beim W3C eingereicht[3] und soll folgende Vorteile bringen: „WS-Addressing decouples address information from the specific transport used by providing a mechanism to place the target, source and other important address information directly within the Web service message.“ (Weerawarana et al. 2005, 39)

Dazu werden zwei Konstrukte in Form eines XML-Dokuments eingeführt, die die entsprechenden Informationen strukturiert beinhalten. Erstens ist dies eine Endpunkt-Referenz (engl. *endpoint references*), die „[...] alle notwendigen Informationen für die Verwendung eines Web-Service-Endpunktes enthält.“ (Dostal et al. 2005, 66) Mittels der Referenz-Eigenschaften (engl. *reference properties*) wird ein Endpunkt um zusätzliche Angaben erweitert, so „dass zwei Endpunkt-Referenzen mit gleicher URI und unterschiedlichen Referenz-Eigenschaften unterschiedliche Ressourcen repräsentieren“ (Dostal et al. 2005, 66). Die Referenz-Parameter (engl. *reference parameters*) beinhalten Details des jeweiligen Aufrufs, um z.B. eine Session-Verwaltung zu implementieren, die auf transportspezifische Mechanismen wie HTTP-Cookies verzichtet. Zweitens werden die sog. *„Message Information Header“* für eine Endpunkt-Referenz eingesetzt, die „diverse Eigenschaften der SOAP-Interaktion bzgl. Routing und Kommunikationspfad [festlegen], wobei hier eine transportunabhängige Spezifikation erfolgt.“ (Dostal et al. 2005, 67) Beispielsweise sind dies die Ziel-Adresse, eine Nachrichten-ID, ein Antwort- oder Fehler-Endpunkt.

4.2 Dienstbeschreibung mittels WSDL

Zur Dienstbeschreibung wird im Web Service Kontext fast ausnahmslos die *Web Service Description Language* (WSDL) eingesetzt, die in Form eines XML-Dokuments einen

[3] `http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/`, verifiziert am 09.01.2006, 16:54 MEZ

Service syntaktisch oder strukturell beschreibt. Ein WSDL-Dokument enthält in maschinenlesbarer Form sämtliche Funktionen und Parameter (z.B. das Format der Ein- bzw. Ausgabenachrichten), um mit einem Web Service zu kommunizieren, ähnlich der IDL bei anderen Middleware-Plattformen wie z.B. CORBA (siehe Kapitel 3.3) (vgl. hierzu Apfel 2004, 21). Dies wird durch die WSDL in einer plattformübergreifenden Weise geleistet, da im Gegensatz zu IDL keine feste Bindung an eine Middleware-Plattform und damit vorbestimmte Interaktionsmechanismen vorliegen. Vielmehr kann ein und derselbe Dienst über verschiedene Protokolle angesprochen werden, was mittels WSDL ausgedrückt werden kann (vgl. Alonso et al. 2004, 166). Jedoch beinhaltet ein WSDL-Dokument keinerlei semantische Informationen, um einen Dienst und seine angebotenen Funktionalitäten inhaltlich zu beschreiben.

WSDL entstand im Jahre 2000 als ein Amalgamat der bis zu diesem Zeitpunkt inkompatiblen Beschreibungssprachen NASSL (Network Application Service Specification) von IBM und SDL (Service Description Language) von Microsoft. Die Version 1.1 von WSDL[4] ist quasi der de facto Standard bei der Beschreibung von Web Services, für den bislang die größte Unterstützung an Software-Tools vorhanden ist. Mittlerweile wird an der Version 2.0 von WSDL gearbeitet[5], die sich jedoch nach Einschätzung von Weerawarana et al. (2005, 126) auf Grund der festen Verankerung von WSDL 1.1 in der Entwickler-Community und den bereits verfügbaren Tools erst langsam durchsetzen wird.

In den folgenden Kapiteln wird zunächst ein genereller Überblick über den Aufbau eines WSDL-Dokuments gegeben, um anschließend auf die Besonderheiten und Unterschiede zwischen den beiden WSDL-Versionen (1.1 und 2.0) einzugehen.

4.2.1 Aufbauprinzipien und Datencodierung bei WSDL-Dokumenten

Die WSDL-Spezifikation weist einen *abstrakten* Teil, der die Funktionalitäten eines Web Services beschreibt und einen *konkreten* Teil auf, der die „technischen Details über den Ort und die Art und Weise, wie ein Dienst angeboten wird [beinhaltet].“ (Dostal et al. 2005, 78) WSDL 1.1 und WSDL 2.0 unterscheiden sich in ihrer syntaktischen Struktur voneinander (siehe Abbildung 4.4). Konzeptuell beibehalten wurde aber bei beiden eine Trennung zwischen dem, „was“ „wie“ und „wo“ angesprochen werden soll: „It separates what the service does (expressed in the form of `<portType>` in WSDL 1.1 and `<interface>` in WSDL 2.0) from how the service is to be interacted with (expressed in the form of `<binding>` in WSDL 1.1 and 2.0) and where the service is offered (expressed in the form of `<port>` in WSDL 1.1 and `<endpoint>` in WSDL 2.0). This separation allows one description of what a service does to be offered in different forms of interaction at different locations.“ (Weerawarana et al. 2005, 110)

[4] `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`, verifiziert am 10.01.2006, 14:33 MEZ
[5] veröffentlicht am 06.01.2006 im Status einer „W3C Candidate Recommendation“, `http://www.w3.org/TR/2006/CR-wsdl20-20060106`, verifiziert am 10.01.2006, 14:40 MEZ

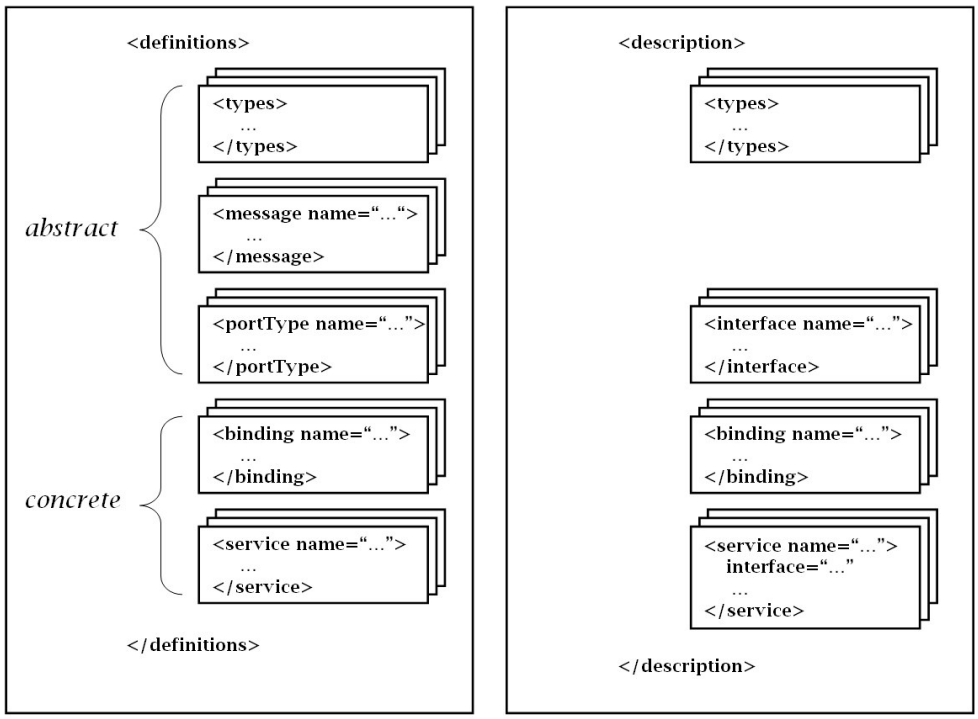

Abbildung 4.4: Syntaktische Struktur eines WSDL-Dokuments (a) WSDL 1.1, (b) WSDL 2.0

Damit die zwischen den Kommunikationspartnern ausgetauschten Daten korrekt interpretiert werden können, muss ein gemeinsames Codierungsschema für Datentypen verwendet werden. WSDL stellt keine eigene solche Beschreibungssprache zur Verfügung, sondern erlaubt die Nutzung verschiedener Codierungssprachen, wie z.B. XML Schema, RelaxNG[6] oder auch andere nicht-XML basierte Sprachen wie MIME[7] (Multipurpose Internet Mail Extensions) oder COBOL copybook[8]. Im Gegensatz zu anderen Middleware-Plattformen wie CORBA, ist das Datentypen-System nicht implizit durch die verwendete Plattform vorgegeben, sondern muss festgelegt werden (vgl. Alonso et al. 2004, 167). Standardmäßig nutzt WSDL das XML Schema Codierungssystem, das einfache Datentypen vorgefertigt bereitstellt und bei Bedarf um nutzerspezifische, komplexe Datentypen (z.B. Strukturen) erweitert werden kann: „[...] most users have utilized XML Schema as the type system language and converted whatever their native type system is to XML Schema. Although that is the most flexible in terms of understandability by consumers of the service (as more people will understand XML Schema), it is not the only way to WSDL." (Weerawarana et al. 2005, 114) Eine XML Schema Definition kann direkt innerhalb des

[6] `http://relaxng.org/spec-20011203.html`, verifiziert am 11.01.2005, 9:50 MEZ
[7] `http://www.ietf.org/rfc/rfc2045.txt`, verifiziert am 11.01.2006, 9:26 MEZ
[8] „Eine Copybook Beschreibung ist eine Codedatei, die von mehreren Cobol-Quellen verwendet wird, um die Struktur eines Datenstroms zu beschreiben. Diese Datenströme werden in der Regel in Dateien persistiert, die nach den verschiedenen Verfahren (EBCDIC, Binär, ASCII, Packed-Decimal, etc.) kodiert werden. Mit Hilfe von Copybook Definitionen werden mit Cobol erstellte Dateien interpretiert bzw. Dateien erstellt, die von Cobol-Anwendungen verarbeitet werden können.", `http://www.softproject.de/de/products/integration/adapter/cpy/`, verifiziert am 11.01.2006, 9:41 MEZ

Elements <types> erfolgen. Nachfolgend wird ein Ausschnitt aus dem WSDL-Dokument für den Amazon-WS[9] wiedergegeben, der den Datentyp bzw. das Objekt „ItemSearch“ beschreibt. Dieses Objekt ist ein komplexes Objekt, da neben primitiven Datentypen, wie Strings o.Ä., weitere in XML Schema definierte Datentypen verwendet werden. In diesem Fall ist es der Typ „ItemSearchRequest“, der wiederum selbst ein komplexes Objekt darstellt.

```
<types>
<xs:element name="ItemSearch">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="SubscriptionId" type="xs:string" minOccurs="0"/>
        <xs:element name="AWSAccessKeyId" type="xs:string" minOccurs="0"/>
        <xs:element name="AssociateTag" type="xs:string" minOccurs="0"/>
        <xs:element name="XMLEscaping" type="xs:string" minOccurs="0"/>
        <xs:element name="Validate" type="xs:string" minOccurs="0"/>
        <xs:element name="Shared" type="tns:ItemSearchRequest" minOccurs="0"/>
        <xs:element name="Request" type="tns:ItemSearchRequest" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
</xs:element>
</types>
```

4.2.2 Elemente eines WSDL 1.1 Dokuments

4.2.2.1 Messages

„Das Nachrichtenelement (<wsdl:message>) dient zum Festlegen der Anfrage-, Antwort- und der Fehlernachrichten, die ein Web Service verarbeitet und die somit zwischen dem Dienstanbieter und dem Dienstnachfrager ausgetauscht werden.“ (Burghardt 2004, 57) Dabei wird auf die Standard-XML Schema Datentypen sowie auf die im Element <types> angegebenen komplexen Datentypen zurückgegriffen. Auf Grund seiner Kompaktheit folgt als Beispiel ein Ausschnitt aus dem WSDL-Dokument des Google-WS[10]. Es werden die Nachrichtenformate für die Operationen „doGoogleSearch“ (bestehend aus primitiven Datentypen) und „doGoogleSearchResponse“ (als komplexer Datentyp) definiert:

```
  <message name="doGoogleSearch">
    <part name="key"            type="xsd:string"/>
    <part name="q"              type="xsd:string"/>
    <part name="start"          type="xsd:int"/>
    <part name="maxResults"     type="xsd:int"/>
    <part name="filter"         type="xsd:boolean"/>
    <part name="restrict"       type="xsd:string"/>
    <part name="safeSearch"     type="xsd:boolean"/>
    <part name="lr"             type="xsd:string"/>
    <part name="ie"             type="xsd:string"/>
    <part name="oe"             type="xsd:string"/>
  </message>

  <message name="doGoogleSearchResponse">
    <part name="return"         type="typens:GoogleSearchResult"/>
  </message>
```

[9] http://webservices.amazon.com/AWSECommerceService/DE/AWSECommerceService.wsdl, verifiziert am 11.01.2006, 14:43 MEZ

[10] http://api.google.com/GoogleSearch.wsdl, verifiziert am 11.01.2006, 9:43 MEZ

4.2.2.2 PortType

Das Element `PortType` beinhaltet die Definition der Operationen (mittels des Kind-Elements `<operation>`), die dem Dienstnutzer zur Verfügung gestellt werden. Dazu werden im Element `<operation>` die Nachrichten, die als Anfrage-, Antwort- und Fehlernachrichten benutzt werden, zusammengefasst. „In WSDL 1.1, operations are named and defined by the messages that the service receives or sends; each operation can send or receive at most one message in each direction.“ (Weerawarana et al. 2005, 116) Es gibt in WSDL 1.1 vier Arten von Operationen, die im sich anschließenden WSDL-Fragment in der Reihenfolge ihrer Nennung illustriert werden (Weerawarana et al. 2005, 116f.):

- **One-way:** Der Service empfängt eine Nachricht ohne eine Antwort zurückzusenden.
- **Request-response:** Der Service empfängt eine Nachricht und sendet eine Antwort.
- **Solicit-response:** Der Service sendet eine Nachricht und erhält eine Antwort.
- **Notification:** Der Service sendet eine Nachricht und erwartet keine Antwort.

```
<portType name="p1">

  <operation name="op1">
     <input message="x:m1"/>
  </operation>

  <operation name="op2">
     <input message="x:m1"/>
     <output message="y:m2"/>
  </operation>

  <operation name="op3">
     <output message="x:m1"/>
     <input message="y:m2"/>
  </operation>

  <operation name="op4">
     <output message="x:m1"/>
  </operation>

</portType>
```

Die Operations-Arten „request-response“ und „solicit-response“ sind nicht zwangsläufig synchrone Operationen und müssen auch nicht an synchrone oder blockierende Transportprotokolle gebunden werden, um z.B. eine sofortige Antwort zu erhalten. Innerhalb von WSDL 1.1 ist dies nicht festgelegt. Die oben genannten Operationen können ebenso über asynchrone Protokolle, wie z.B. SMTP, versandt werden (vgl. Weerawarana et al. 2005, 117).

4.2.2.3 Bindings

Mittels des Elements `<binding>` wird das Nachrichtenformat definiert und dem abstrakten PortType ein konkretes Transportprotokoll (z.B. SOAP, HTTP GET/POST oder MIME) zugeordnet. Mit einer Bindung z.B. an das SOAP-Protokoll wird festgelegt, wie aus den

Eingabe- oder Ausgabenachrichten des <message>-Elements ein SOAP Envelope zu erstellen ist. Um SOAP Interaktionstypen zu modellieren, kann bei der Bindung an das SOAP-Protokoll zwischen zwei Nachrichtenformaten (vgl. Kapitel 4.1.2.1) gewählt werden (Weerawarana et al. 2005, 119):

- **Document style:**
  „Document style basically means that all the parts of the <message> are directly inserted into the SOAP envelope as children of the <Body> element.
- **RPC style:**
  RPC style basically means that all the parts of the <message> are wrapped in some outer element representing the RPC. Then that resulting single wrapper element is inserted as the single child of SOAP's <Body> element."

Zum Nachrichtenformat gehört auch die Festlegung der zu verwendenden Codierung, um eine Nachricht von/in XML zu de/codieren bzw. zu de/serialisieren. Dazu bietet sich entweder das Vorgehen *„literal"* oder *„encoded"* (= SOAP encoding) an (vgl. Kapitel 4.1.2.2). „**Literal encoding** takes the WSDL types defined in XML Schemas and „literally" uses those definitions to represent the XML content of messages. In other words, the abstract WSDL types also become concrete types. **SOAP encoding**, on the other hand, takes the XML Schema definitions as abstract entities, and translates them into XML using SOAP encoding rules defined as part of SOAP 1.2." (Anm.: Hervorhebungen durch den Verfasser, Alonso et al. 2004, 169) Zur Illustration folgt ein Fragment aus dem WSDL-Dokument des Google-WS, der als Nachrichtenformat RPC und als Codierungsformat „encoded" verwendet:

```
<binding name="GoogleSearchBinding" type="typens:GoogleSearchPort">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>

    <operation name="doGetCachedPage">
      <soap:operation soapAction="urn:GoogleSearchAction"/>
      <input>
        <soap:body use="encoded" namespace="urn:GoogleSearch"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded" namespace="urn:GoogleSearch"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
   </operation>
</binding>
```

WSDL 1.1 ist neben SOAP die wohl am weitesten verbreitete Web Service Spezifikation überhaupt. Sie bietet eine große Flexibilität, um einen Dienst zu beschreiben. Jedoch kann gerade diese Flexibilität zu Inkompatibilitäten führen, wenn Implementierungen verschiedener Hersteller zusammen arbeiten. Daher wurden durch die WS-I (vgl. Kapitel 3.4) mit dem WS-I Basic Profile (aktuell in der Version 1.1, siehe Ballinger et al. 2004) einige Möglichkeiten von WSDL eingeschränkt, um so größtmögliche Interoperabilität zu gewährleisten (basierend auf Nutzererfahrungen und bekannten Interoperabilitätsproblemen). In seinem Artikel stellt Butek (2005) die verschiedenen Kombinationen der

Nachrichtenstile (RPC/encoded, RPC/literal, document/encoded, document/literal und zusätzlich das von Microsoft eingeführte document/literal wrapped) einander gegenüber und benennt die Vor- und Nachteile der Kombinationsmöglichkeiten. Das WS-I Basic Profile in der Version 1.1 empfiehlt aus Gründen der Interoperabilität die Codierungsart „literal“ und ein dokumentenorientiertes Nachrichtenformat (document/literal).

4.2.2.4 Ports und Services

Ports oder Endpunkte (engl. *endpoints*) führen die Bindings mit konkreten Netzwerkadressen (URLs) zusammen, unter denen die Implementation eines abstrakten PortTypes zu erreichen ist. Ein Port wird mit dem Element `<port>` definiert. Dem Element `<service>` werden ein oder mehrere `port`-Elemente als Kind-Elemente hinzugefügt und bilden somit die von einem Service bereitgestellten Dienste. Im Folgenden wird dies wieder anhand eines Ausschnitts aus dem WSDL-Dokument des Google-WS verdeutlicht:

```
<service name="GoogleSearchService">
    <port name="GoogleSearchPort" binding="typens:GoogleSearchBinding">
     <soap:address location="http://api.google.com/search/beta2"/>
   </port>
</service>
```

4.2.3 Elemente eines WSDL 2.0 Dokuments

Auf Grund der im Einsatz von WSDL 1.1 gemachten Erfahrungen, zeigte sich die Notwendigkeit zur Weiterentwicklung und Vereinfachung von WSDL (syntaktische Struktur von WSDL 2.0 siehe Abbildung 4.4 (b)). Im Folgenden werden nur die wichtigsten Änderungen benannt (für eine ausführliche Beschreibung von WSDL 2.0 siehe Kapitel 5 Dostal et al. 2005).

In WSDL 2.0 wurde das Element `<message>` komplett gestrichen und stattdessen werden die auszutauschenden Informationen über die in einem WSDL-Dokument global verfügbaren XML Schema Definitionen beschrieben (siehe Weerawarana et al. 2005, 122ff.). Als Nachrichtenstil wurde standardmäßig „document/literal“ festgelegt, das WSDL 1.1 Element `<portType>` wurde in `<interface>` und das Element `<port>` in `<endpoint>` umbenannt:

```
<interface name="ExampleInt">
    <operation name="op1">
      <input element="x:input-element"/>
      <output element="y:output-element"/>
    </operation>
</interface>
```

Die vier Operationen zur Steuerung des Nachrichtenflusses (one-way, request-response, solicit-response, notification) wurden durch das systematische Konzept der „Message Exchange Patterns“ (MEPs) ersetzt (siehe Dostal et al. 2005, 86ff.), die in der aktuellen

WSDL 2.0 Version acht MEPs umfassen. Damit kann beispielsweise die Reihenfolge und Häufigkeit von Nachrichten festgelegt werden, die während der Dienstnutzung ausgetauscht werden und nach welchen Regeln Fehlercodes gesendet werden sollen. Zudem wurde die Definition des `<service>`-Elements dahingehend abgeändert, dass dieses Element in WSDL 2.0 eine Menge von Endpunkten beinhaltet, die ein einziges, gemeinsames Interface implementieren. Das erhöht die Übersichtlichkeit. So kann beispielsweise ein Interface über zwei verschiedene Bindungen an zwei Endpunkten angesprochen werden (Beispiel entnommen aus Weerawarana et al. 2005, 125):

```
<service name="ExampleService" interface="x:ExampleInt">
  <endpoint name="b1" binding="y:Binding1" address="hhtp://foo.com/bar1"/>
  <endpoint name="b2" binding="z:Binding2" address="mailto:bar1.service@foo.com"/>
</service>
```

4.3 Dienstverzeichnis

Verzeichnisdienste sind ein zentraler Bestandteil einer SOA (vgl. Kapitel 3.2). Durch standardisierte Schnittstellen ermöglichen sie eine lose Kopplung von Anwendungen, indem Dienste dynamisch gesucht, gefunden und genutzt werden. Zwei im Ansatz grundsätzlich verschiedene Spezifikationen für Verzeichnisdienste gibt es: Die *„Web Service Inspection Language“* (WS-Inspection) und die *„Universal Description, Discovery and Integration“* (UDDI) Spezifikation, die in diesem Kapitel vorgestellt werden. Im Moment scheinen beide Technologien sich noch nicht großflächig verbreitet zu haben, was sich aber mit der Weiterentwicklung der SOA Thematik, insbesondere von Web Services, in Zukunft ändern wird.

4.3.1 WS-Inspection

Statt eines großen, zentralen Verzeichnisses aller Dienste (wie bei UDDI), setzt WS-Inspection auf viele, kleine dezentrale Verzeichnisse. Damit wird versucht, so Dostal et al. (2005, 105), „eines der Hauptprobleme, die momentan mit den Implementierungen von UDDI-Verzeichnissen einhergehen, nämlich die ungenügende Moderation der Verzeichnisse und die damit verbundene Unzuverlässigkeit der Suchergebnisse“, zu umgehen.

Dazu wird auf dem Webserver des Anbieters ein XML-Dokument mit dem (zwingend vorgeschriebenen Namen) *inspection.wsil* angelegt, das eine Liste der Web Services und einen Verweis z.B. auf ein WSDL-Dokument des jeweiligen Services beinhaltet. Ein Nutzer kann über das HTTP-Protokoll auf dieses Dokument zugreifen. Ein *inspection.wsil*-Dokument ist so aufgebaut, dass es aus einem `<inspection>`-Element besteht, das beliebig viele `<service>`-Elemente (zur Definition der Dienste und deren technische Beschreibung z.B. Speicherort des WSDL-Dokuments) oder `<link>`-Elemente (Verweise auf externe Datenquellen, z.B. andere WS-Inspection-Dokumente oder UDDI-Verzeichnisse) enthalten darf.

WS-Inspection-Dokumente können auf Grund des <link>-Elements hierarchisch angeordnet werden, um z.B. ein Kategorisierungssystem aufzubauen. Außerdem können diese Dokumente dynamisch erzeugt werden, z.B. aus einem internen UDDI-Verzeichnis oder anderen Verzeichnissen. (vgl. Dostal et al. 2005, 105ff.)

4.3.2 UDDI

Innerhalb eines UDDI-Verzeichnisses (engl. *UDDI repository* oder *registry*) können Anbieter ihren Dienst mit einer Dienstbeschreibung (z.B. in Form eines WSDL-Dokuments) in einer Datenbank hinterlegen, damit ein potentieller Nutzer diesen Dienst und eine Anleitung zu dessen Nutzung finden kann. Dies kann entweder ein menschlicher Nutzer (z.B. mittels einer webbasierten Anwendung) oder ein maschineller Nutzer (z.B. ein Programm, das automatisch weitere Funktionalitäten einbinden möchte) sein. Die UDDI-Spezifikation entstand wieder durch Kooperation mehrerer Firmen (v.a. IBM, Microsoft und Ariba) und wird mittlerweile von der OASIS-Organisation (Organization for the Advancement of Structured Information Standards)[11] gepflegt. Zurzeit aktuell ist die Version 3 des UDDI-Standards[12].

UDDI besteht aus zwei großen Komponenten: Einem UDDI-XML-Schema, das das Datenmodell definiert und einer UDDI-API, die selbst als Web Service implementiert ist und das SOAP-Protokoll nutzt. Um das Datenmodell besser zu verstehen, wird eine Analogie zu Telefonbüchern hergestellt: In den *„white pages“* (vgl. Adressbuch) sind die Organisationen, ihre Kontaktinformationen (z.B. Telefon- oder E-Mail Adressen) und die von der jeweiligen Organisation angebotenen Dienste hinterlegt. In den *„yellow pages“* (vgl. „Gelbe Seiten“) findet eine Kategorisierung der Dienste statt, z.B. nach Branchen, wobei entweder eine standardisierte oder eine nutzerdefinierte Taxonomie zum Einsatz kommt. Die *„green pages“* halten Informationen vor, wie ein Dienst anzusprechen ist, z.B. in Form einer Referenz auf ein WSDL-Dokument (vgl. Alonso et al. 2004, 175).

Die vier Hauptentitäten des UDDI-XML-Schemas (siehe Abbildung 4.5) und ihre Funktionen sind:

- **businessEntity:** Diese Datenstruktur beinhaltet Informationen zu Organisationen oder Unternehmen, die Web Services anbieten.
- **businessService:** Hier werden die von einer „businessEntity“ bereitgestellten Dienste zusammengefasst. Dabei können verschiedene Dienste gruppiert werden (ein Gehaltsrechner, ein Reisebuchungs-Service) und jeweils an verschiedene Übertragungsprotokolle gebunden werden. Ein <businessEntity>-Element kann mehrere <businessService>-Elemente beinhalten, jedoch gehört ein „businessService-Element“ ausschließlich zu einer „businessEntity“.

[11] http://www.oasis-open.org, verifiziert am 05.02.2006 17:20 MEZ
[12] Version 3.0 unter http://uddi.org/pubs/uddi-v3.00-published-20020719.htm als Spezifikation veröffentlicht; aktuelle Version 3.0.2 siehe http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

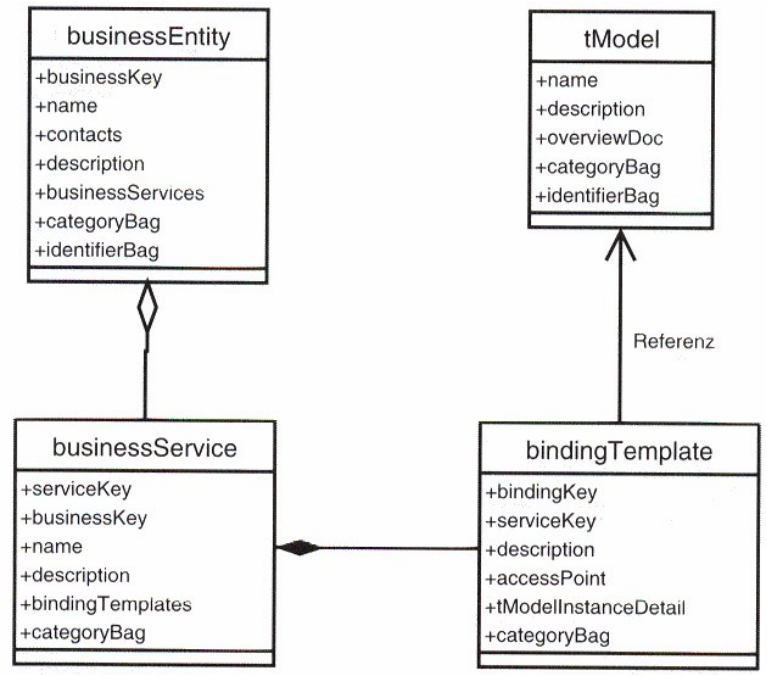

Abbildung 4.5: UDDI Datenmodell (Dostal et al. 2005, 111)

- **bindingTemplate:** Die technischen Details zur Nutzung eines Dienstes sind hier gespeichert. Das ist einerseits die Adresse, unter der ein Dienst aufgerufen werden kann. Andererseits umfasst dies auch ein oder mehrere Dokumente, die die eindeutige technische Beschreibung des Dienstes beinhalten (das sog. „tModel“) (vgl. Dostal et al. 2005, 111). Ein „bindingTemplate“ gehört zu genau einem „businessService“.
- **tModel:** Der Begriff steht für „technical model“ und dient als generischer Datencontainer für jegliche Art von Spezifikation. Das kann beispielsweise ein WSDL-Dokument oder eine andere XML-Spezifikation zur Dienstbeschreibung sein; die UDDI Spezifikation gibt hier keine Sprache vor. Diese Dienst-Spezifikation ist als Referenz meist auf dem Server des Dienstanbieters hinterlegt. Anders als die zuvor genannten Entitäten steht das „tModel“ mit keiner anderen Entität in einer Eltern-Kind-Beziehung (vgl. Alonso et al. 2004, 176). Das ermöglicht, „Web Services auf technischer Ebene miteinander zu vergleichen und automatisiert zu prüfen, wie ein Web Service angesprochen werden muss und welche Spezifikationen und Konstrukte er verwendet.“ (Dostal et al. 2005, 117)

Zur Interaktion mit einem UDDI-Verzeichnis werden mehrere APIs eingesetzt, die sich an verschiedene Nutzerrollen und Aufgaben richten. Die beiden wohl wichtigsten APIs sind verantwortlich für die Suche (mittels der **UDDI Inquiry API**) und das Veröffentlichen eines Services in einem UDDI-Verzeichnis (via der **UDDI Publication API**). Daneben gibt es noch weitere APIs, so z.B. eine die für eine Synchronisation zwischen verschiedenen UDDI-Verzeichnissen sorgt (vgl. hierzu Alonso et al. 2004, 180f.).

Bei UDDI handelt es sich um eine Technologie, die noch relativ jung und noch nicht so weit verbreitet ist. Bislang scheint es, laut (Alonso et al. 2004, 182), dass sich die Suche nach Diensten in einem strukturierten UDDI-Verzeichnis schwieriger gestaltet als im unstrukturierten Web (z.B. durch Missverständnisse beim Befüllen der UDDI Datenstrukturen). Insgesamt werfen technische Probleme eher geringere Fragen auf, als die rechtlichen und wirtschaftlichen Aspekte des Einsatzes von Web Services: „Wie ist die Qualität der Web Services? Wie erfolgt die Abrechnung eines genutzten Web Service? Wer trägt die Verantwortung für einen Web Service?“ (Dostal et al. 2005, 108) Im Zusammenhang mit UDDI muss die Umsetzung einer UBR (UDDI Business Registry) gesehen werden, die ein internetweites, kostenloses und öffentliches UDDI-Verzeichnis ist. Die vier öffentlichen UDDI-Registries von Microsoft, SAP, IBM und NTT-COM werden zu einer UBR zusammengeschlossen, die somit einen branchenunabhängigen, öffentlichen Marktplatz darstellt und Dienstanbieter und -nutzer zusammen führt (vgl. Dostal et al. 2005, 124ff.).

Jedoch wurde die öffentliche UBR, bestehend aus den jeweiligen UDDI-Verzeichnissen der beteiligten Firmen, zu Jahresbeginn von Microsoft, SAP und IBM eingestellt. Als Begründung wurde von IBM z.B. angegeben, dass dies nur ein „Proof of concept“ darstellte, der mittlerweile Eingang in kommerzielle Produkte gefunden habe und weiterentwickelt wird (vgl. `http://www-306.ibm.com/software/solutions/webservices/uddi/`, verifiziert am 22.03.2006, 20:00 MEZ). Eine Einschätzung zu diesem Vorgang liefert Jason Bloomberg (zitiert nach einem Artikel von Paul Krill[13]):

> "Basically, the UBR is a relic of an earlier vision for UDDI. The original vision for UDDI was as a standard that would help companies conduct business with each other in an automated fashion. The idea was that companies could publish how they wanted to interact, and other companies could find that information and use it to establish a relationship [...] Needless to say, this isn't how companies do business – there's always a human element to establishing a relationship. As a result, the UBR served as little more than an interoperability reference implementation. Now that UDDI has become more of a metadata management standard for SOA, there's little need for the UBR anymore [...]"

4.4 Zusammenfassung

Gegenstand dieses Kapitels waren die grundlegenden Techniken und Protokolle zum Einsatz von Web Services. Der Aufbau und die Möglichkeiten des Kommunikationsprotokolls SOAP wurden vorgestellt und zugleich auf einen Schwachpunkt der SOAP-Spezifikation und deren Kompensation mittels des Standards WS-Addressing eingegangen.

[13] `http://www.infoworld.com/article/05/12/16/HNuddishut_1.html`, verifiziert am 22.03.2006, 20:05 MEZ

Zur Dienstbeschreibung werden WSDL-Dokumente eingesetzt, wobei die Unterschiede in den WSDL-Versionen (1.1 und 2.0) für die Werkzeuge zur WSDL-Verarbeitung (vgl. Kapitel 7.1) von besonderer Bedeutung sein werden. Zum Suchen eines Dienstes wurde schwerpunktmäßig das UDDI-Protokoll vorgestellt.

Als zusammenhängendes Beispiel wird das Nutzen eines Web Services angeführt, der zu einer gegebenen IP-Adresse das Land ermittelt, dem die IP-Adresse zugeordnet ist (adaptiertes Beispiel entnommen aus `http://www.webservicex.net/geoipservice.asmx?WSDL`, verifiziert am 22.03.2006, 18:30 MEZ). Dieser Web Service wird durch folgendes WSDL-Dokument beschrieben:

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:s="http://www.w3.org/2001/XMLSchema"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns="http://www.webservicex.net"
                  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
                  targetNamespace="http://www.webservicex.net" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://www.webservicex.net">
      <s:complexType name="GeoIP">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="IP" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="CountryCode" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="CountryName" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="ReturnCode" type="s:int" />
          <s:element minOccurs="0" maxOccurs="1" name="ReturnCodeDetails" type="s:string" />
        </s:sequence>
      </s:complexType>
      <s:element name="GetGeoIP">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="IPAddress" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetGeoIPResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetGeoIPResult" type="tns:GeoIP" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GeoIP" nillable="true" type="tns:GeoIP" />
    </s:schema>
  </wsdl:types>

  <wsdl:message name="GetGeoIPSoapIn">
    <wsdl:part name="parameters" element="tns:GetGeoIP" />
  </wsdl:message>
  <wsdl:message name="GetGeoIPSoapOut">
    <wsdl:part name="parameters" element="tns:GetGeoIPResponse" />
  </wsdl:message>

 <wsdl:operation name="GetGeoIP">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">GeoIPService -
             GetGeoIP enables you to easily look up countries by IP addresses
      </documentation>
      <wsdl:input message="tns:GetGeoIPSoapIn" />
      <wsdl:output message="tns:GetGeoIPSoapOut" />
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="GeoIPServiceSoap" type="tns:GeoIPServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <wsdl:operation name="GetGeoIP">
      <soap:operation soapAction="http://www.webservicex.net/GetGeoIP" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>


  <wsdl:service name="GeoIPService">
    <wsdl:port name="GeoIPServiceSoap" binding="tns:GeoIPServiceSoap">
      <soap:address location="http://www.webservicex.net/geoipservice.asmx" />
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

Es wird ein Service mit dem Namen „GeoIPService“ bereitgestellt, der über das Binding „GeoIPServiceSOAP“ Nachrichten über SOAP im document/literal-Style austauscht. Die Operation, die dabei angesprochen wird, heißt „GetGeoIP“. Als Eingabeparameter wird ein String-Wert erwartet („GetGeoIP“), der die IP-Adresse enthält, deren Ort ermittelt werden soll. Der Ausgabeparameter ist ein komplexer Datentyp vom Type „GeoIP“. Er ist aus mehreren String-Datentypen und einem Integer-Wert zusammengesetzt.

Im Rahmen der Anfrage an diesen Web Service wird eine SOAP-Nachricht erstellt und über HTTP versandt, was im Original-Quelltext wie folgt ausschaut[14]:

```
POST /geoipservice.asmx HTTP/1.1
Host: www.webservicex.net
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.webservicex.net/GetGeoIP"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetGeoIP xmlns="http://www.webservicex.net">
      <IPAddress>194.97.206.218</IPAddress>
    </GetGeoIP>
  </soap:Body>
</soap:Envelope>
```

Die über HTTP empfangene Antwort ist wiederum eine SOAP-Nachricht und sieht folgendermaßen aus[15]:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetGeoIPResponse xmlns="http://www.webservicex.net">
      <GetGeoIPResult>
        <IP>194.97.206.218</IP>
        <CountryCode>DE</CountryCode>
        <CountryName>Germany</CountryName>
        <ReturnCode>1</ReturnCode>
        <ReturnCodeDetails>Record Found</ReturnCodeDetails>
      </GetGeoIPResult>
    </GetGeoIPResponse>
  </soap:Body>
</soap:Envelope>
```

Bei der Beschreibung von Diensten ist insbesondere hervorzuheben, dass WSDL keine Möglichkeit vorsieht, semantische Informationen zur Dienstbeschreibung zu liefern. Ein

[14] http://www.webservicex.net/geoipservice.asmx?op=GetGeoIP, verifiziert am 22.03.2006, 18:22 MEZ

[15] a.a.O.

WSDL-Dokument fungiert vielmehr ausschließlich als technische Anleitung zur Dienstnutzung. Eine deskriptive Beschreibung der Fähigkeiten eines Dienstes und dessen Leistungen sind damit nicht möglich. Es gibt zwar Werkzeuge, die WSDL-Dokumente auf Ähnlichkeit prüfen. Dabei wird nur überprüft, ob die beschriebenen Dienste auf dieselben Schnittstellen (d.h. Endpunkte) zurückgreifen. Eine Prüfung auf semantische Identität von Web Services und ihren verrichteten Funktionen findet nicht statt.

Dieses Problem wird besonders beim Einsatz einer Service-Registry, z.B. durch UDDI deutlich. Es stellt sich die Frage, wie ein Nutzer einen bestimmten Dienst ermitteln kann. Ein mögliches Vorgehen ist die Schlagwortsuche in einer vom Dienstanbieter erstellten und gepflegten Dienstbeschreibung. Die zum Beschreiben verwendete Terminologie ist im Idealfall für alle von *einem* Dienstanbieter zur Verfügung gestellten Dienste systematisch aufgebaut (evtl. sogar in Form eines kontrollierten Vokabulars), jedoch besteht zwischen den verschiedenen Dienstanbietern kein gemeinsamer Standard. Wie soll nun ein Nutzer mittels einfacher Schlagwortsuche einen Dienst finden können, der mit einem synonymen Schlagwort versehen worden ist? An dieser Stelle stößt man auf klassische Fragestellungen im Bereich Information Retrieval wie z.B. Indexerstellung oder automatische Erweiterung von Anfragen mit ähnlichen Suchtermen, um z.B. die Treffermenge zu erhöhen oder aber der Einsatz von Ontologien, die einen Gegenstandsbereich systematisch gliedern und als Grundlage für das Modellieren von semantischen Konzepten genutzt werden können.

Auch bei den im folgenden Kapitel vorgestellten Protokollen zum Modellieren von Geschäftsprozessen wird davon ausgegangen, dass die eingesetzten Dienste zuvor ermittelt wurden und nur noch technisch miteinander gekoppelt werden.

5 Geschäftsprozess-Modellierung mittels Web Services

Um die Bereiche Workflow-Management und Web Services zu verknüpfen, gibt es bereits mehrere Ansätze zur Standardisierung, die jedoch bei verschiedenen Gremien (W3C, OASIS u.a.) eingereicht wurden und – auch inhaltlich – miteinander konkurrieren. Der wohl aussichtsreichste Kandidat eines Standards ist die Business Process Execution Language for Web Services (BPEL4WS oder auch WS-BPEL), da dieser von fast allen Großunternehmen unterstützt wird und daher in Kapitel 5.2 ausführlicher beschrieben wird (vgl. Dostal et al. 2005, 198). Bevor diese Spezifikation näher erläutert wird, werden zunächst die Begriffe und Konzepte der Orchestrierung und Choreographie von Web Services beschrieben, die grundlegend für das Verständnis bei der Geschäftsprozess-Modellierung mit Web Services sind.

5.1 Orchestrierung und Choreographie von Web Services

Martens (2004, 16) beschreibt, dass Unternehmen ihre bisher verteilt ablaufenden Geschäftsprozesse zu *einem verteilten Geschäftsprozess* kombinieren wollen. „Ein *verteilter Geschäftsprozess* besteht aus lokalen Geschäftsprozessen, die räumlich und/oder organisatorisch getrennt ablaufen und lediglich über Kommunikationskanäle verbunden sind." (Martens 2004, 16) Als Motive für das Zusammenführen benennt Martens zum einen die interne Aufteilung eines Unternehmens (z.B. Niederlassungen an verschiedenen Orten) und zum anderen das Entstehen von *virtuellen Unternehmen*. Das ist ein zeitlich begrenztes Zweckbündnis von rechtlich unabhängigen Unternehmen, das einem Dritten gegenüber wie ein einheitliches Unternehmen auftritt und auf eine zeitlich befristete Kooperation ausgelegt ist.

Beim Einsatz eines monolithischen, zentralisierten Prozessmanagements bei virtuellen Unternehmen ist besonders problematisch, dass ein globales Prozessmodell bei jeder lokalen Änderung eines Partnerprozesses bearbeitet werden muss. Dies führt in der Praxis dazu, dass „die einzelnen beteiligten Organisationen die Autonomie über ihre lokalen Prozesse verlieren." (Martens 2004, 19) Außerdem würde es, so Martens, die beteiligten Partner zwingen, ihre lokalen Prozessmodelle offen zu legen, was im Sinne des Schutzes von Betriebsgeheimnissen oftmals nicht gewünscht ist. Zudem sind die Einrichtung und der Unterhalt einer zentralen Komponente zeit- und kostenintensiv, was bei den auf eine zeitlich beschränkte Zusammenarbeit ausgerichteten virtuellen Unternehmen ein besonders gewichtiges Gegenargument darstellt. Eine dezentrale Struktur scheint daher geeigneter zu sein, indem lokale Geschäftsprozesse in Verantwortung der jeweiligen

Partner bleiben und eine globale Kooperation sich auf das Nötigste beschränkt. Diesen Ansatz können Web Services leisten (vgl. Martens 2004, 19f.).

Ein Unternehmen kann, basierend auf seiner lokalen IT- und Organisations-Infrastruktur, mit Hilfe von Web Services (lokale) Geschäftsprozesse bereitstellen und veröffentlichen. Einen verteilten Geschäftsprozess erhält man, wenn mehrere dieser lokalen Geschäftsprozesse zu einem Gesamtprozess verknüpft werden: „[. . .] a client can itself be exposed as a Web service, and as such it can be used as a component by other, higher-level (composite) services." (Alonso et al. 2004, 247) Dieses Vorgehen ermöglicht eine Reduzierung der Komplexität, da ein komplexerer Dienst inkrementell aus weniger komplexen Bausteinen zusammengefügt werden kann.

Eine Integration von Geschäftsprozessen ließe sich auch über Workflow-Management-Systeme oder andere EAI-Systeme (Enterprise Application Integration) erledigen. Meist lassen sich diese Systeme sehr flexibel und generisch erweitern, indem für den jeweiligen speziellen Anwendungsfall sog. „Adapter" (engl. *adapter* oder *wrapper*) geschrieben werden müssen. Der dafür benötigte Entwicklungsaufwand steigt mit der Anzahl der zu integrierenden Systeme (vgl. Kapitel 3.5). Web Services hingegen bieten Standardschnittstellen und -protokolle, die eine effizientere und werkzeuggestützte Integration erlauben, bis hin zum Zusammenstellen von Geschäftsprozess-Abläufen mittels Drag&Drop am Bildschirm über graphische Editoren (vgl. Alonso et al. 2004, 252ff.). Die Komposition von Web Services beinhaltet zwei Aspekte, die in den folgenden beiden Kapiteln erläutert werden.

5.1.1 Choreographie

Einzelne Operationen von Web Services können mittels WSDL beschrieben werden, nicht jedoch die korrekte und erwartete Abfolge der Nachrichten. „Eine Choreographie beschreibt die Aufgaben und das Zusammenspiel *mehrerer* Prozesse unter dem Aspekt der Zusammenarbeit." (Dostal et al. 2005, 202) Hierbei wird die zulässige Nachrichtenabfolge zwischen den Kommunikationspartnern festgelegt (engl. *coordination protocol*) und jedem Kommunikationspartner (Anwendung, Web Service, Nutzer etc.) eine Rolle zugewiesen (Kunde, Anbieter, Bank). Beispielsweise soll erst eine Kaufbestätigung an einen Kunden geschickt werden, wenn dieser ein Anbieterangebot akzeptiert hat – und nicht schon nach Abschicken seiner Anfrage zur Angebotserstellung an den Anbieter! Eine graphische Modellierung kann beispielsweise mit Zustands-, Sequenz- oder Aktivitätsdiagrammen (siehe Abbildung 5.1) erfolgen (siehe hierzu auch Alonso et al. 2004, Kapitel 7).

„Die Choreographie stellt eine Art öffentliche Schnittstelle dar. Sie ist nicht als Prozess ausführbar. Sie beschreibt den Nachrichtenaustausch aus einer öffentlichen Sicht, macht aber keine Angaben über die interne Prozesslogik der Teilnehmer." (Reichert und Stoll

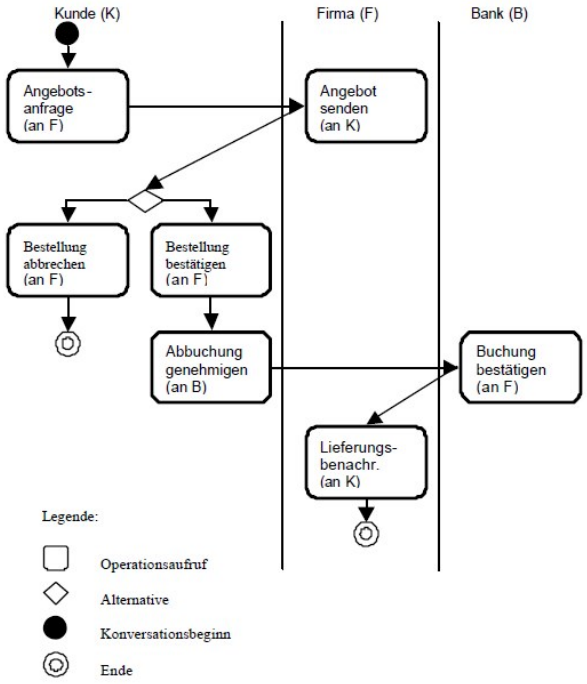

Abbildung 5.1: Modellierung einer Web Service Choreographie mittels eines Aktivitätsdiagramms (Reichert und Stoll 2004, 27)

2004, 27) Die konkrete Umsetzung eines Geschäftsprozesses bleibt somit ein „Betriebsgeheimnis“ des Dienstanbieters. Zur Koordination (engl. *coordination*) von Web Service Aktivitäten ist es erforderlich, dass beispielsweise eine Aktivität einer konkreten Prozess-Instanz zugeordnet werden kann. Dazu kann das Protokoll WS-Coordination herangezogen werden (vgl. Kapitel 6.2.2).

5.1.2 Orchestrierung

„Orchestrierung beschreibt die Geschäftsprozesslogik aus Sicht eines Teilnehmers, d.h. die Reihenfolge und Ausführungsbedingungen der Aufrufe externer oder firmeneigener Web Services.“ (Reichert und Stoll 2004, 28) Dies gleicht der Ablaufsteuerung eines Geschäftsprozesses durch ein Workflow-Management-System. Orchestrierung wird auch als (engl.) *„service composition“* bezeichnet.

Das im vorangegangenen Kapitel angeführte Beispiel wird zur Illustration fortgeführt: Der Web Service „Anfrage entgegennehmen“ der Firma F kann aus firmeninterner Sicht weiter aufgespaltet werden (siehe Abbildung 5.2). Dabei sind die internen Geschäftsprozesse der Firma gestrichelt dargestellt; sie sind nach außen hin nicht sichtbar (so z.B. der Preisvergleich bei mehreren Lieferanten) und stellen sozusagen die „Betriebsgeheimnisse“ dar. Diese Abläufe können firmenintern geändert werden, ohne dass die Kooperationspartner darüber informiert werden müssen. Das spiegelt den Grundgedanken der losen Kopplung wider (vgl. Kapitel 3.1).

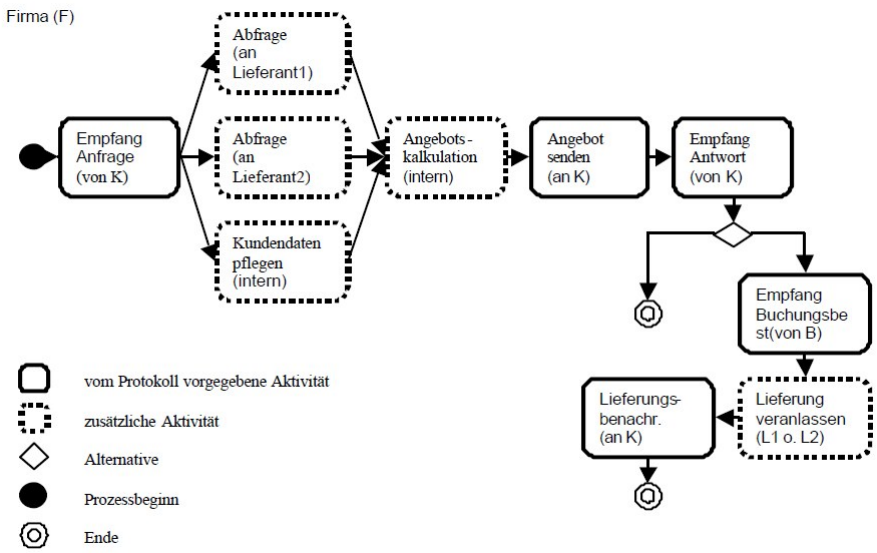

Abbildung 5.2: Orchestrierung von Web Services mehrerer Partner (Reichert und Stoll 2004, 28)

5.2 WS-BPEL als Sprache zur Geschäftsprozess-Modellierung (Orchestrierung)

WS-BPEL ist eine imperative Programmiersprache, die Geschäftsprozesse mit Hilfe von standardisierten Beschreibungselementen modelliert, die in einem XML-Dokument abgespeichert werden. Die BPEL4WS-Spezifikation[1] in der Version 1.1 entstand im Jahre 2003 durch Vereinigung der bislang konkurrierenden Spezifikationen von IBM (Web Service Flow Language, WSFL) und Microsoft (XLang, d.h. XML Language) und wurde zur weiteren Standardisierung beim Gremium OASIS eingereicht. Mittlerweile wird an der aktuellen Version 2.0 gearbeitet, die in Angleichung an die anderen Web Service Spezifikationen den Namen WS-BPEL[2] trägt. Zum zeitlichen Entwicklungs- und Entstehungsverlauf von WS-BPEL siehe zur Muehlen et al. (2004, 5ff.) und Dostal et al. (2005, 9f.).

WS-BPEL nutzt die Standards XML Schema 1.0, XPath 1.0 (z.B. um Knoten des WS-BPEL-Dokuments zu adressieren), WSDL 1.1 (zur Beschreibung, welche Dienste genutzt werden sollen und wie der gesamte WS-BPEL Prozess wiederum selbst als Web Service genutzt werden kann) und WS-Addressing (zur Beschreibung von Service-Endpunkten) (vgl. Dostal et al. 2005, 207).

Ein WS-BPEL-Prozess kann in zweierlei Weise genutzt werden. Einerseits können *abstrakte Prozesse* beschrieben werden (engl. *business protocols*), andererseits auch *ausführbare Prozesse* (engl. *executable processes).* „Business protocols or message exchange protocols

[1] `http://www.oasis-open.org/committees/download.php/2046/BPEL%20V1-1%20May%205%202003%20Final.pdf`, verifiziert am 18.01.2006, 11:17 MEZ

[2] aktueller Entwurf unter `http://www.oasis-open.org/committees/download.php/16181/-%20WS-BPEL%202.0%20Specification%20Draft%20September%201%20%28DK%20Updates%20in%2013.3%20%20%2013.4%2C%20Jan%2011%29.doc`, verifiziert am 17.01.2006, 11:22 MEZ

describe the externally visible interactions between business partners without necessarily exposing the internal business logic of the individual partners. In contrast, executable processes contain the partner's business logic behind the external protocol." (Weerawarana et al. 2005, 321) Ein Unternehmen kann z.B. seine öffentlichen Geschäftsprozesse mittels abstrakter Prozessbeschreibungen veröffentlichen, um somit den unternehmensexternen Zugriff darauf zu ermöglichen und gleichzeitig die interne Implementierung „geheim" zu halten (Weerawarana et al. 2005, 321).

Im Folgenden wird eine Auswahl der wichtigsten WS-BPEL Sprachelemente präsentiert. Für eine umfangreichere Darstellung der Neuerungen von WS-BPEL Version 2.0 gegenüber Version 1.1 siehe Präsentation von König (2005).

5.2.1 Kommunikation und Bindung

Die Kommunikationspartner (andere Geschäftsprozesse, Web Services oder andere Clients) eines WS-BPEL-Prozesses werden als Partner bezeichnet und mit Hilfe des Elements `<partnerLinks>` definiert. „Ein BPEL-Prozess benutzt Dienste von Partnern und bietet selbst Dienste nach außen an." (Marks 2005, 62) Ein Beispiel (Dostal et al. 2005, 220):

```
<partnerLinks>
  <partnerLink
      name="bestellen"
      partnerLinkType="GrossistHandelLink"
      myRole="Kaeufer"
      partnerRole="Kunde" />
</partnerLinks>
```

Innerhalb der Definition des `<partnerLink>`-Elements wird festgelegt, welche Rolle der eigene Prozess und welche der externe Partner übernimmt. Werden beide Rollen definiert, handelt es sich um eine synchrone Request-Response-Kommunikation; wird nur eine Rolle definiert, läuft die Kommunikation asynchron ab, z.B. als Request bei alleiniger Angabe von „myRole". Damit letztlich ein Web Service angesprochen werden kann, muss festgelegt werden, welcher konkrete Web Service zum Ausführen des Prozesses eingesetzt werden soll. Dazu verweist das WS-BPEL Attribut `partnerLinkType` auf einen abstrakten `<PortType>` eines WSDL-Dokuments.

Eine konkrete Bindung an einen Service-Endpunkt findet nicht während der „Programmierung" mittels BPEL statt, sondern zum Ausführungszeitpunkt in der Laufzeitumgebung. Sie geschieht unter Zuhilfenahme von Endpunkt-Referenzen der WS-Addressing-Spezifikation. Hierbei wird der `PartnerLink`-Rolle eines Prozesspartners ein Endpunkt zugeordnet, was zu vier verschiedenen Zeitpunkten durchgeführt werden kann (vgl. Weerawarana et al. 2005, 324):

- **zum Modellierungszeitpunkt:** Das ist die unflexibelste Art der Bindung, da jede Installation und jede Prozessinstanz auf diese Endpunkt-Referenz zugreift.

- **zum Deployment-Zeitpunkt:** Im Prozessmodel werden zwar keine konkreten Endpunkte definiert, jedoch werden sie beim Deployment festgelegt. Alle Prozessinstanzen nutzen daher dieselben Endpunkte.
- **dynamisch zur Laufzeit:** Zur Laufzeit werden geeignete Dienste gesucht. Diese müssen zusätzlich Kriterien erfüllen, die als Anforderungen im <partnerLink>-Element festgelegt wurden (z.B. Service-Qualität oder Kosten).
- **dynamisches Binden mit Übermitteln von Endpunkt-Referenzen:** Die Endpunkt-Referenz ist ein Ergebnis einer vorangegangenen Interaktion mit einem Partner.

5.2.2 Datenfluss und Aktivitäten

Die in einem WS-BPEL-Dokument enthaltenen Sprachelemente besitzen einen Gültigkeitsbereich, der mittels des Elements <scope> definiert werden kann. Die Gültigkeitsbereiche können ineinander verschachtelt werden. Hierbei gilt wie in Java der Grundsatz, dass lokale Variablen die globalen Variablen überdecken. Das Wurzelelement eines WS-BPEL-Dokuments stellt die Definition eines <process>-Elements dar (siehe unten, entnommen aus Dostal et al. 2005, 208):

```
<process>
  <!--process definiert implizit den äußersten Gültigkeitsbereich-->
  <scope>
     <!-- Aktivitäten und/oder andere Gültigkeitsbereiche -->
  </scope>
  <!-- weitere Aktivitäten und/oder andere Gültigkeitsbereiche -->
</process>
```

Die Daten eines Prozesses werden über das Element <variable> gespeichert. „Der Datentyp einer Variablen kann entweder ein *WSDL message type*, ein *XML Schema simple type* oder ein *XML Schema element* sein“. Um Variablen oder Partner Links einen Wert zuzuweisen (engl. *assign*), wird das <assign>-Element genutzt (vgl. Dostal et al. 2005, 210ff.):

```
<variables>
  <variable name="maxprice" type"xsd:int"/>
  <variable name="address" messageType=""/>
  <variable name="name" element=""/>
</variables>

<assign>
  <copy>
     <from>1000</from>
     <tovariable="maxprice">
  </copy>
</assign>

<invoke name="ErsterBerechnungsschritt"
        partner="meinAddierer" portType="addiererNS:Addierer"
        operation="addiere"
        inputVariable="ersteSummanden" outputVariable="erstesErgebnis">
</invoke>
```

Die Aktivitäten innerhalb eines WS-BPEL-Dokuments werden in Basisaktivitäten (d.h. elementare Aktivitäten) oder strukturierte Aktivitäten (d.h. aus anderen Aktivitäten zusammengesetzt) unterteilt. Die *Basisaktivitäten* kommunizieren beispielsweise mit Web Services (`<invoke>`) oder mit Partnern (`<receive> und <reply>`; bei `<receive>` blockiert der Prozess solange, bis eine für ihn bestimmte SOAP-Nachricht eintrifft). *Strukturierte Aktivitäten* beschreiben entweder den sequentiellen Ablauf von Aktivitäten (`<sequence>`, `<switch>`[3], `while`) oder einen parallelen oder synchronisierten Ablauf (`<flow>`) (Beispiel entnommen aus Dostal et al. 2005, 216f.):

```
<sequence>
    <!--activityA-->
    <!--activityB-->
    <!--activityX-->
    <!--activityY-->
</sequence>

<whilecondition="bpws:getVariableData('counter')<10">
    <!--someactivities-->
</while>

<switch>
    <casecondition="amout>0">
          <invoke ...>
    </case>
    <otherwise>
          <receive ... />
    </otherwise>
</switch>

<flow>
    <invoke ... >
    </invoke>
    <sequence>
          <receive ... />
          <invoke ... />
          </invoke>
    </sequence>
</flow>
```

5.2.3 Kontrollfluss und Fehlerbehandlung

Mittels der Aktivitäten `<sequence>` und `<flow>` lassen sich sehr viele Geschäftsprozesse modellieren, jedoch nicht alle, was von Dostal et al. (2005, 217) mit folgendem Beispiel verdeutlicht wird: „Ein *flow* bestehend aus drei geordneten *sequences* ist beispielsweise in einem XML-Dokument sehr einfach zu beschreiben, gibt es jedoch zusätzlich einige Abhängigkeiten in der Ausführungsreihenfolge zwischen Aktivitäten aus den drei *sequences* untereinander, so ist eine Modellierung nur über *sequences* und *flows* in einigen Fällen schlicht nicht möglich.“ Zur Lösung dieses Problems wurden in WS-BPEL die sog. `<links>` eingeführt, die man sich z.B. als Kante in einem Graph vorstellen kann. Sie ermöglichen das Verbinden von Elementen über eine vorgegebene Hierarchie hinaus, wie sie z.B. durch das Festlegen im WS-BPEL XML-Dokument beschrieben wurde. Dieser

[3] Dieses Element ist in der aktuellen Version 1.1 enthalten. Der Entwurf von Version 2.0 zu WS-BPEL ersetzt dieses Konstrukt durch eine `<if> ...<then> ...<else>`-Anweisung

Vorgang wird auch mit *Synchronisation von Aktivitäten* bezeichnet. „Der Fluss der beiden Aktivitäten verläuft nun von der Aktivität mit dem Element *source* zur Aktivität mit dem Element *target*.“ (Dostal et al. 2005, 218) Dostal et al. führen hierfür folgendes Beispiel an: Die im vorherigen Abschnitt vorgestellte sequentielle Abarbeitung innerhalb des `<sequence>`-Elements wurde nachfolgend mittels einer Flow-Aktivität (d.h. parallele Verarbeitung) und Links zur Einschränkung der Abarbeitungsreihenfolge nachgebildet. Das ist zwar möglich, aber kein empfohlenes Vorgehen, da die Übersichtlichkeit für evtl. Änderungen darunter stark leidet. Bis auf Ausnahmefälle sollte daher auf das `<link>`-Element verzichtet werden.

```
<flow>
  <links>
      <link name="AvorB">
      <link name="XvorY">
  </links>

  <invoke name="A">
      <source linkName="AvorB">
  </invoke>

  <invoke name="B"
      <target linkName="AvorB">
  </invoke>

  <invoke name="X">
      <source linkName="XvorY">
  </invoke>

  <invoke name="Y">
      <target linkName="XvorY">
  </invoke>
</flow>
```

Mit der Ausnahmebehandlung (engl. *exception handling*) wird auf Ereignisse oder Fehler reagiert, die vom erwarteten oder gewünschten Ausführungszustand abweichen. Serverausfälle oder Systemfehler sind Beispiele dafür. Alonso et al. (2004, 273ff.) führen drei grundlegende Muster zur Ausnahmebehandlung an („flow-based“, „try-catch-throw approach“ und „rule-based“), von denen das ebenfalls aus der Programmiersprache Java bekannte Muster „try-catch-throw“ in BPEL eingesetzt wird. Der Vorteil dieser Herangehensweise liegt in der strikten Trennung zwischen „normal“ und im Fehlerfall ausgeführten Code, was zu einer besseren Übersicht während der Programmierung führt. Außerdem besteht die Möglichkeit, trotz eingetretener Ausnahme, den Prozessablauf weiter definiert fortzuführen.

In WS-BPEL entspricht der Aufbau des `<faultHandler>`-Elements im Wesentlichen dem catch-Block. Tritt eine Ausnahme auf, wird erst im lokalen Gültigkeitsbereich nach einem `<faultHandler>`-Element gesucht. „Ist in diesem Bereich keines definiert, so wird von unten hierarchisch in den umgebenden Gültigkeitsbereichen gesucht. Sollte selbst im Gültigkeitsbereich des Elements `<process>` keine Fehlerbehandlung durchgeführt werden, wird der Prozess terminiert.“ (Dostal et al. 2005, 228) Im unten aufgeführten Beispiel (aus Dostal et al. 2005, 228) wird bei einem aufgetretenen Fehler während der

Datenbankverbindung eine Aktion *kompensiert*, ansonsten wird der gesamte Prozess terminiert.

```
<faulthandler>
  <catchfaultName="NoDatabaseConnectionFault">
  <compensate/>
  </catch>
  <catchAll>
      <terminate/>
  </catchAll>
</faulthandler>
```

Eine *Kompensation*, d.h. ein Rückgängigmachen von ausgeführten Aktivitäten, wird über sog. `<compensationHandler>` abgewickelt. Sie können explizit nach dem erfolgreichen Durchführen einer Aktivität aufgerufen werden (`<compensate>`-Aktivität) oder durch einen `<faultHandler>`. Die Herangehensweisen zur Realisierung von Transaktions- und Kompensationsmechanismen werden in Kapitel 6.2 aufgezeigt.

5.3 Zusammenfassung

Betrachtet man Web Services als Komponenten, eröffnen sich dadurch neue Möglichkeiten zum Modellieren von Geschäftsprozessen. Statt innerhalb eines Geschäftsprozesses detailliert Funktionalität zu implementieren, wird auf einen vorhandenen Web Service zurückgegriffen. Diese komponentenbasierte Sichtweise der Software-Entwicklung wird auch mit *Programming in the large* bezeichnet, bei der als Voraussetzung für eine erfolgreiche Interaktion eine genaue Schnittstellendefinition zwischen den verschiedenen Komponenten erfolgen muss. Durch die standardisierte Sprache WSDL zur Schnittstellenbeschreibung ist diese Voraussetzung im Web Service Umfeld gegeben.

Das bloße Verfügbarmachen der einzelnen Module via Schnittstellen, in diesem Fall über Web Services, reicht zum Erstellen von komplexen Geschäftsprozessen nicht aus. Es werden Mechanismen benötigt, die z.B. den Ablauf koordinieren und eine Fehlerbehandlung ermöglichen. Die Sprache WS-BPEL ermöglicht eine solche strukturierte Beschreibung und wurde mit ihren Grundkonzepten in diesem Kapitel vorgestellt. Sie ermöglicht einem Programmierer, sozusagen aus der Vogelperspektive verschiedene Web Services zu kombinieren und zu einem großen, komplexen Geschäftsprozess zusammen zu stellen. Er muss dabei nicht über die jeweilige Implementierung eines einzelnen Services Bescheid wissen (was dem *Programming in the small* entspricht), sondern kann anhand der Schnittstellenbeschreibung komponentenbasiert arbeiten.

Die Modellierung von Geschäftsprozessen stellt besondere Anforderungen an die verwendeten Techniken hinsichtlich Sicherheit (z.B. im unternehmensübergreifenden Einsatz) und der Abwicklung von Transaktionen. Im Web Services-Bereich wurden für diese geschäftskritischen Basisfunktionalitäten bei der Modellierung von Geschäftsprozessen bereits Standards entwickelt, die im folgenden Kapitel beschrieben werden.

6 Sicherheit und Transaktionen bei Web Services

Mittels Web Services lassen sich auf einfache Art und Weise verteilte Anwendungssysteme miteinander verbinden. Vor allem bei der Nutzung von Diensten über Unternehmensgrenzen hinweg stellen sich schnell Fragen nach Sicherheitszielen wie Vertraulichkeit, Datenintegrität, Zurechenbarkeit und Zugriffskontrolle, die zwischen den Kommunikationspartnern zu gewährleisten sind. In den vorangegangenen Kapiteln wurden die technischen Grundlagen zur Umsetzung von Web Services mit Standards und Protokollen beschrieben, die nur eine „unsichere“ Datenübermittlung beherrschen. Beispielsweise ist beim Versenden von SOAP-Nachrichten via HTTP ein Mitlesen der Nachricht im Klartext möglich. Daher wird in Kapitel 6.1 die Umsetzung der zuvor benannten Sicherheitsziele bei der Nutzung von Web Services aufgezeigt.

Besonders bei Geschäftsprozessen, die aus mehreren Web Services aufgebaut sind, spielt die Abwicklung von Transaktionen eine große Rolle, um einen bei allen Beteiligten konsistenten Bearbeitungszustand sicherzustellen. Die grundlegenden Prinzipien und die sich herausbildenden Standards werden in Kapitel 6.2 vorgestellt.

6.1 Sicherheit bei Web Services

Sicherheitsziele wurden vor allem im Umfeld der Netzwerksicherheit definiert und gelten genauso für den Einsatz von Web Services: *Vertraulichkeit* heißt, dass eine Nachricht von Dritten nicht unberechtigt gelesen werden kann, was klassischerweise von Verschlüsselungsverfahren bewerkstelligt wird. Spricht man von *Datenintegrität*, so meint man damit die unveränderte Übertragung der Daten zwischen Sender und Empfänger, wobei jeglicher Verlust, jede Modifikation und jegliche Ergänzung der übertragenen Informationen erkannt werden muss. Mittels Maßnahmen zur *Zugriffskontrolle* soll sichergestellt werden, dass nur berechtigte Nutzer einen Dienst verwenden. Zudem soll die Nutzung eines Web Services später nicht abgestritten werden können bzw. versandte Nachrichten sollen einem konkreten Urheber zugeordnet werden können, was mit *Zurechenbarkeit* umschrieben wird und wiederum durch Verfahren der digitalen Signatur umgesetzt wird (vgl. Dostal et al. 2005, 80).

Bei den Verschlüsselungsverfahren wird zwischen symmetrischen, asymmetrischen und hybriden Verfahren unterschieden. Bei den *symmetrischen* Verfahren benutzen Sender und Empfänger denselben Schlüssel. Hauptvorteil ist dabei eine höhere Geschwindigkeit der dafür eingesetzten Algorithmen (z.B. Data Encryption Standard – DES, Triple-DES oder AES – Advanced Encryption Standard). Jedoch – das ist der Hauptnachteil und

die Hauptschwierigkeit – muss der verwendete Schlüssel zwischen den Kommunikationspartnern zuvor ausgetauscht werden. *Asymmetrische* oder *public-key-Verfahren* verwenden ein sog. „Schlüsselpaar“, das aus einem privaten und öffentlichen Schlüssel besteht. Sender und Empfänger verwenden jeweils separate Schlüssel zum (De-)Codieren der Nachrichten. Die dafür eingesetzten Algorithmen (z.B. RSA - benannt nach den Entwicklern Ronald L. Rivest, Adi Shamir und Leonard Adleman) sind jedoch nicht so schnell wie die symmetrischen Verfahren. Daher werden in der Praxis oftmals *hybride* Verfahren eingesetzt, bei denen die Kommunikation symmetrisch verschlüsselt abläuft und der dafür erforderliche Schlüssel zuvor mittels asymmetrischer Verfahren ausgetauscht worden ist (vgl. Burghardt 2004, 81f.).

Besonders im Rahmen der *digitalen Signatur* werden asymmetrische Verfahren eingesetzt, um das unveränderte Eintreffen von Daten sicherzustellen – nicht jedoch eine vertrauliche Übermittlung (siehe Abbildung 6.1). Zuerst wird vom Absender über die zu versendende Nachricht eine Prüfsumme bzw. ein Secure Hash-Wert erzeugt (z.B. mittels eines Secure Hash-Verfahrens wie SHA [Secure Hash Algorithm]), der eine Nachricht eindeutig identifiziert. Mit seinem privaten Schlüssel verschlüsselt der Absender den Hash-Wert und fügt ihn als so genannte Signatur der Original-Nachricht hinzu. Signatur und Original-Nachricht werden zusammen über den Kommunikationskanal übertragen.

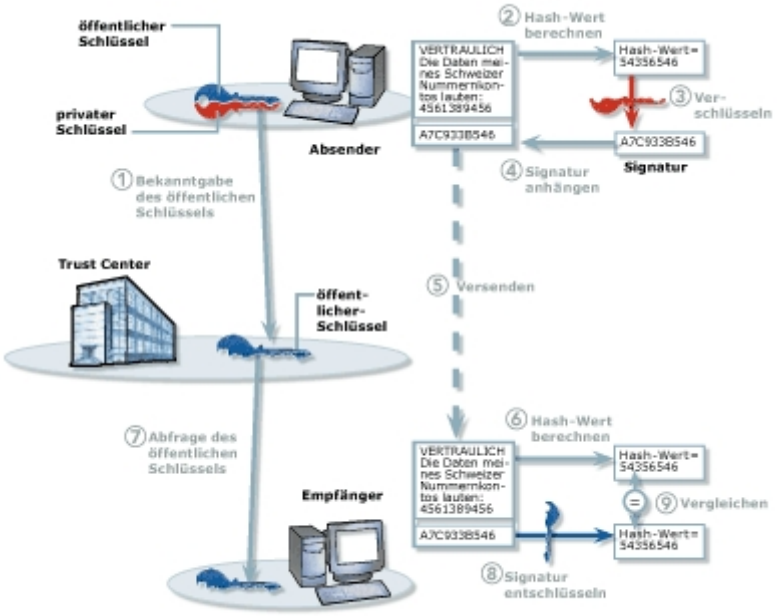

Abbildung 6.1: Abläufe zum Erstellen und Testen einer Digitalen Signatur, Quelle: `http://www.microsoft.com/switzerland/security/de/privat/images/1_1_2_4_digitale_signatur.gif`, verifiziert am 20.03.2006, 16:20 MEZ

Der Empfänger nutzt den öffentlichen Schlüssel des Senders, um aus der Signatur den ursprünglichen Secure Hash-Wert zu ermitteln. Gleichzeitig berechnet er einen eigenen Hash-Wert über die erhaltene Nachricht und vergleicht beide Hash-Werte miteinander:

Stimmen beide Prüfsummen überein, ist die Nachricht nicht verändert worden. Erhielt der Empfänger den öffentlichen Schlüssel aus einer vertrauenswürdigen Quelle (z.B. durch ein Trust-Center), kann er bei übereinstimmenden Prüfsummen sogar die Echtheit des Absenders dieser Nachricht annehmen (vgl. Burghardt 2004, 83f.).

Bereits existierende Ansätze zur Nachrichtensicherheit im Kontext von HTTP spielen sich überwiegend auf der Ebene der Transportschicht des ISO/OSI-Referenzmodells ab (vgl. Burghardt 2004, 86ff.). Dabei wird eine sichere „Punkt-zu-Punkt-Verbindung“ zwischen Sender und Empfänger erstellt, damit Nachrichten geschützt vor unbefugtem Mitlesen oder Modifizieren ausgetauscht werden können. „Da es sich aber um einen echten Kanal zwischen zwei Beteiligten handelt, ist die Lösung für eine echte verteilte Anwendung, wie sie in einer SOA realisiert wird, meist ungeeignet. Im Web-Services-Umfeld ist es spätestens bei der Nutzung von Intermediären meist nicht mehr möglich, die Daten sicher zu übertragen.“ (Dostal et al. 2005, 170) Eventuell vorhandene Intermediäre könnten nämlich auf die gesamte SOAP-Nachricht unberechtigterweise zugreifen oder sie verändern, ohne dass dies bemerkt würde. Dies ist besonders wichtig, da Web Services eine sog. „Ende-zu-Ende-Sicherheit“ propagieren, d.h. ein durchgängig vorhandener Sicherheitskontext zwischen dem Dienstanbieter und -nutzer, sowie aller evtl. daran beteiligter Intermediäre. Sollen zudem nur bestimmte Nachrichtenteile geschützt werden und im Zuge der Sicherstellung von Zurechenbarkeit eine Art Absendebeweis geführt werden, sind bloße tranportschichtgebundene Maßnahmen wie z.B. TLS (Transport Layer Security) oder SSL (Secure Socket Layer) im Rahmen von Web Services nicht ausreichend. Techniken, die den besonderen Anforderungen im Web Service Umfeld gerecht werden, werden exemplarisch in den folgenden beiden Kapiteln vorgestellt.

6.1.1 XML Digital Signatures und XML-Encryption

Das Web Service Basisprotokoll SOAP nutzt durchgängig XML-Dokumente zur Datenübertragung. Dadurch lassen sich Inhalte und beschreibende XML-Elemente einfach voneinander trennen, um beispielsweise ausschließlich die Nutzdaten oder nur bestimmte Dokumentbereiche bis hin zu einem einzelnen XML-Element zu verschlüsseln oder zu signieren.

Eine Besonderheit von XML-Dokumenten stellt die „Formatfreiheit“ dar (vgl. Dostal et al. 2005, 177): Leerzeichen, die ausschließlich der Formatierung dienen (und nicht innerhalb von Elementgrenzen liegen), können ohne Informationsverlust weggelassen werden. Aus diesem Grund können Dokumente mit dem gleichen Inhalt eine vollkommen andere Formatierung aufweisen. Während der Verarbeitung von XML-Dokumenten könnte evtl. eine XML-konforme Umformatierung eintreten, bei der z.B. zusätzliche Leerzeichen eingefügt werden. Dies kann bei der Prüfsummenberechnung anschließend Probleme verursachen, wenn aus diesem Grund beispielsweise eine andere Signatur errechnet wurde, obwohl das Ausgangsdokument und das transformierte Dokument inhaltlich identisch

sind, jedoch unterschiedlich formatiert sind. Aus diesem Grund wird mittels Canonical XML[1] aus jedem XML-Dokument eine eindeutige, normalisierte Repräsentation erzeugt, „[...] sodass inhaltlich gleiche Dokumente mit unterschiedlicher Formatierung, Codierung und so weiter auf dasselbe kanonische XML-Dokument abgebildet werden." (Dostal et al. 2005, 177)

Zum Einsatz von digitalen Signaturen im XML-Kontext kann der W3C-Standard *„XML-Signature Syntax and Processing"*[2] herangezogen werden. Dostal et al. (2005, 176) nehmen an, dass sich diese Spezifikation in Zukunft erfolgreich durchsetzt, da andere Standards (z.B. WS-Security) diesen Standard benutzen. Er gewährleistet die Überprüfung von Integrität und Zurechenbarkeit einer Nachricht (siehe Beispiel eines signierten SOAP-Dokuments in Dostal et al. 2005, 179).

Soll die Vertraulichkeit von XML-Daten gewahrt werden, kommt die ebenfalls vom W3C gepflegte Spezifikation *„XML Encryption"*[3] zum Tragen. Neben der Verschlüsselung eines Gesamtdokuments wird in dieser Spezifikation festgelegt, dass einzelne Teile mit verschiedenen Schlüsseln chiffriert werden können, um innerhalb eines XML-Dokuments ein abschnittsweises Festlegen von Vertraulichkeit zu ermöglichen (z.B. für den eigentlichen Empfänger oder speziell für die bei der Kommunikation beteiligten Intermediäre). XML-Encryption und XML Digital Signature dienen dazu, sichere „Ende-zu-Ende-Verbindungen" mit durchgängigem Sicherheitskontext zwischen einem Dienstnutzer und einem Dienstanbieter herzustellen (Burghardt 2004, 88ff.).

6.1.2 WS-Security

Der Standard *WS-Security* verknüpft bewährte Verfahren und formuliert ein standardisiertes Vorgehen bei der Nutzung von bspw. XML Digital Signature und XML-Encryption. WS-Security soll als Basis für alle weiteren Sicherheitsstandards für Web Services dienen, weswegen sämtliche Erweiterungen wie z.B. WS-Trust darauf basieren (siehe Abbildung 6.2). Hauptaugenmerk bei dieser Spezifikation liegt auf einer sicheren Übertragung auf Nachrichtenebene. Ein standardisiertes Vorgehen bei der Verarbeitung soll ein problemloses Zusammenspiel zwischen verschiedenen Beteiligten gewährleisten, so z.B. dass Intermediäre die für sie bestimmten Teile signieren können. Zu diesem Zweck führt die Spezifikation WS-Security neue Elemente für den Header von SOAP-Dokumenten ein.

[1] im Status einer W3C Recommendation in der aktuellen Version vom 15.03.2001; `http://www.w3.org/TR/2001/REC-xml-c14n-20010315`, verifiziert am 10.02.2006, 11:10 MEZ

[2] Das aktuell gültige Dokument in der Version vom 12.02.2005 hat den Status einer W3C Recommendation; `http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/`, verifiziert am 10.02.2006, 11:24 MEZ. Eine Java-API wird im Zuge des Java Community Process erarbeitet mittels eines Java Specification Request (JSR 105), vgl. `http://jcp.org/en/jsr/detail?id=105`, verifiziert am 10.02.2006, 12:21 MEZ

[3] Innerhalb der W3C Working Group „XML Encryption" werden verschiedene Spezifikationen rund um das Thema Verschlüsseln von XML-Daten erstellt (vgl. Liste mit Deliverables unter `http://www.w3.org/Encryption/2001/`, verifiziert am 10.02.2006, 11:32 MEZ. Eine Java-API wird im Rahmen des Java Community Process erarbeitet mit dem Java Specification Request (JSR 106), vgl. `http://jcp.org/en/jsr/detail?id=106`, verifiziert am 10.02.2006, 11:37 MEZ

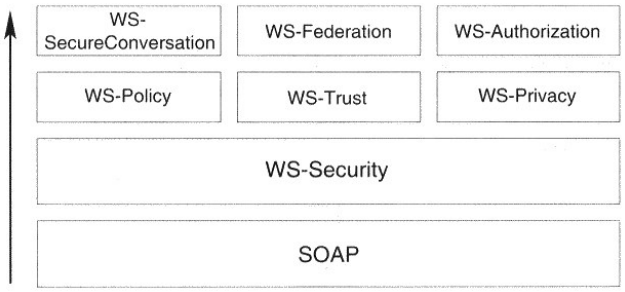

Abbildung 6.2: Erweiterungen von WS-Security (Dostal et al. 2005, 184)

Mögliche Erweiterungen von WS-Security sind (vgl. Dostal et al. 2005, 185ff.):

- **WS-Policy** zur Formulierung von Sicherheitsanforderungen und Richtlinien, die zur Interaktion mit einem Dienst erfüllt werden müssen[4];
- **WS-Trust** zum standardisierten, initialen Austausch von sicherheitsrelevanten und geheimen Daten (z.B. kryptographische Schlüssel)[5];
- **WS-SecureConversation** zum Etablieren eines Sicherheitskontextes zwischen den Kommunikationspartnern. Statt für jeden einzelnen Nachrichtenversand eine Authentifikation durchzuführen, was bei großen Nachrichtenmengen viel Zeit erforderte, wird diese zu Beginn *einmal* erledigt. Anschließend wird ein Sitzungsschlüssel erzeugt, der von beiden Seiten zur Verschlüsselung benutzt wird, um von jetzt an Daten über den Sicherheitskontext miteinander auszutauschen[6].
- **WS-Privacy** zum Beschreiben der Wünsche und Forderungen von Web Services, wenn Daten vertraulich verarbeitet werden sollen. Dabei soll durch die Dienste mittels dieser Spezifikation nachgewiesen werden, dass sie sich selbst an entsprechende Policies (z.B. definiert durch WS-Policy) halten und mit der Einhaltung einverstanden sind.
- **WS-Federation** zur unternehmensübergreifenden Einrichtung einer Vertrauensdomäne. Diese Spezifikation legt fest, wie verfahren wird, wenn die beiden Partner unterschiedliche Standards, z.B. in Form von Sicherheits-Tokens, verwenden. Praktische Ziele sind unter anderem die Umsetzung von Single-Sign-on und eine Abbildung von identischen Identitäten.
- **WS-Authorization** zur Festlegung von Zugriffsrechten und deren Kontrolle bei Web Services.

[4] Aktuelle Spezifikation in der Version vom 01.09.2004 siehe `ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf`, verifiziert am 10.02.2006, 12:01 MEZ
[5] Aktuelle Spezifikation in der Version vom 01.02.2005 siehe `ftp://www6.software.ibm.com/software/developer/library/ws-trust.pdf`, verifiziert am 10.02.2006, 12:04 MEZ
[6] Aktuelle Spezifikation in der Version vom 01.02.2005 siehe `ftp://www6.software.ibm.com/software/developer/library/ws-secureconversation.pdf`, verifiziert am 10.02.2006, 12:05 MEZ

6.2 Transaktionen bei Web Services

Sollen aus mehreren Web Services zusammengesetzt komplette Geschäftsprozesse abgebildet und realisiert werden, muss zum Erreichen eines konsistenten Ziels eine sog. „Transaktionssicherheit“ bestehen. Eine Transaktion ist eine Menge von Operationen, „[. . .] die das zugrunde liegende Informationssystem [oder Datenbanksystem o.Ä.] von *einem* konsistenten Zustand in einen anderen, *ebenfalls* konsistenten Zustand überführt.“ (Dostal et al. 2005, 233)

Folgendes Beispiel, in dem Transaktionen auftreten, führt Burghardt (2004, 109) an: Ein Seminardienstleister fungiert als Dienstanbieter. Die Buchung eines Seminars erfolgt mittels eines Web Services, wobei noch Zusatzleistungen wie Transport und Hotelunterkunft durch den Dienstnutzer gebucht werden können. Ein Dienstnutzer befragt den Web Service des Seminardienstleisters und erhält eine Auskunft über Verfügbarkeit oder die Möglichkeit zur Buchung. Der Dienstanbieter nutzt für die Buchung von Hotels und Transportmöglichkeiten wiederum Web Services anderer Anbieter. Die Buchung eines Seminars mit Zusatzleistungen kann jedoch nur erfolgreich abgeschlossen werden, wenn alle daran beteiligten Einzelaktivitäten erfolgreich ausgeführt werden. Hierbei handelt es sich um eine globale oder verteilte Transaktion (aus Sicht des Dienstnutzers), die aus drei lokalen Subtransaktionen (aus Dienstanbietersicht) besteht.

Service-oriented Architectures stellen besondere Herausforderungen an die Durchführung von Transaktionen, da beispielsweise Nachrichten von Kommunikationspartnern auf Grund von Problemen mit der Netzwerkverbindung oder Fehlern innerhalb der Anwendung, die den Web Service bereitstellt, jederzeit auftreten können. Im Folgenden werden zuerst existierende Konzepte zur Transaktionsabwicklung vorgestellt, um diese anschließend mit den sich entwickelnden Standards im Web Service Umfeld abzugleichen.

6.2.1 Grundlegende Konzepte

6.2.1.1 ACID-Paradigma

Ursprünglich für das Datenbank-Umfeld entwickelt, werden Transaktions-Konzepte überwiegend danach bewertet, welche der vier ACID-Eigenschaften sie erfüllen (vgl. Burghardt 2004, 108):

- **Atomicity:** Alle zu einer Transaktion zugehörigen Einzeloperationen sind als logische Einheit anzusehen und dürfen nur in ihrer Gesamtheit ausgeführt werden. Kann eine Teiloperation nicht durchgeführt werden, muss die komplette Transaktion rückabgewickelt werden.

- **Consistency:** Jede Einzeloperation einer Transaktion führt von einem konsistenten Ausgangszustand in einen konsistenten Endzustand. Wurden alle Teiloperationen erfolgreich durchgeführt, kann als Schlussfolgerung eine konsistente Durchführung der Gesamtoperation garantiert werden.
- **Isolation:** „Andere Transaktionen, die zeitgleich die gleiche Ressource beziehungsweise die gleiche Einzeloperation nutzen, beeinflussen sich nicht gegenseitig und haben somit untereinander keine Wechselwirkungen."
- **Durability:** Am Ende einer erfolgreichen Transaktion werden die Zustandsänderungen persistent gespeichert und gehen so bei nachfolgenden Fehlern von anderen Transaktionen nicht verloren.

6.2.1.2 Verteilte Transaktionen und Zwei-Phasen-Commit

Verteilte Transaktionen sind generell bei Nutzung von kombinierten Web Services anzutreffen, wobei die Erfüllung aller zuvor genannten ACID-Bedingungen nicht immer erforderlich oder möglich ist. Zwei Rollen kann man bei verteilten Transaktionen unterscheiden: Ein **Teilnehmer** (oder Client) führt Einzeloperationen einer Transaktion aus. Ein **Koordinator** (oder Server oder Transaktions-Manager) überwacht die Erzeugung neuer Transaktionen oder den Beitritt von Teilnehmern zu bereits laufenden Transaktionen. Meist wird der Koordinator durch ein Transaktions-Framework bereitgestellt, so dass ein Programmierer lediglich für eine Implementierung der Teilnehmer sorgen und diese für die Teilnahme an der Transaktion anmelden muss.

Bei verteilten Transaktionen – im Gegensatz zu lokalen Transaktionen in einer Datenbank – kann die Erfüllung der ACID-Bedingungen sogar nachteilig sein. Sobald eine Teiloperation fehlschlägt, müssen alle anderen Teiloperationen einer Transaktion rückgängig gemacht werden (engl. *rollback*). Dadurch wurden bereits durchgeführte Arbeitsschritte umsonst abgewickelt, was verlorene Arbeit bedeutet, und das Rollback kann bei vielen zuvor getätigten Teilschritten sehr aufwändig sein (Atomicity-Bedingung). Ferner kann es passieren, dass eine Vielzahl von Ressourcen exklusiv für den Zugriff eines Teilnehmers gesperrt wurde (Isolation-Bedingung) und auf Grund von Netzwerkproblemen eine Entsperrung nicht eintrifft, so dass andere Teilnehmer vergeblich warten.

Im Kontext von verteilten Transaktionen soll sichergestellt werden, dass alle Teilnehmer dem Ergebnis der Transaktion zustimmen, wozu das sog. „Zwei-Phasen-Commit" (2PC, engl. *two-phase-commit*) eingesetzt wird. In der *Abstimmungsphase* fordert der Transaktions-Manager alle beteiligten Dienste auf, ihre Teiloperationen zu beenden. Sie bestätigen entweder die „Prepare-"Nachricht oder falls Konflikte oder Fehler auftraten, wird eine „Cancel-"Nachricht gesendet. In der *Ergebnisphase* entscheidet der Transaktions-Manager auf Basis der Rückmeldung aller Teilnehmer, ob die Transaktion vollständig durchgeführt wird, oder abgebrochen wird. Bei atomaren Transaktionen würde eine „Abort-"Nachricht an die Teilnehmer versandt, um ihren jeweiligen Teil rückgängig zu

machen. Bei Nicht-atomaren Transaktionen kann der Transaktions-Manager entscheiden, ob der fehlerbehaftete Dienst für die Transaktion relevant war oder nicht. Im oben genannten Beispiel könnte ein Seminar und ein Hotel im gewünschten Zeitraum vorhanden sein, jedoch ist keine Organisation des Transports möglich. Der Transaktions-Manager entscheidet nun, ob dennoch eine „Commit-"Nachricht an die fehlerfreien Dienste geschickt wird oder nicht (vgl. Dostal et al. 2005, 234ff.).

6.2.2 WS-Coordination

Nachfolgend werden nun die Komponenten des *Web Service Transaction Frameworks* (WSTF) erläutert. Diese Spezifikationsfamilie umfasst neben der WS-Coordination-Spezifikation auch die Spezifikationen WS-AtomicTransaction (für kurz laufende Transaktionen; typischerweise umgesetzt mit dem 2PC-Protokoll) und WS-BusinessActivity (für lang laufende, verteilte Transaktionen). Sie wird von den Firmen IBM, Microsoft und BEA unterstützt und gilt auf Grund der Unterstützung dieser Industriegrößen als aussichtsreichster Kandidat für eine breite Akzeptanz[7]. Zur Problematik einer nicht-standardisierten Spezifikation (Urheberrecht, mögliche Lizenzgebühren etc.) siehe Dostal et al. (2005, 242f.).

In der Spezifikation WS-Coordination[8] werden Mechanismen definiert, um neue Transaktionen zu erstellen und bereits laufenden Transaktionen beizutreten. Besonders hervorzuheben ist die Erweiterbarkeit dieser Spezifikation, die die Basis für weitere Spezifikationen wie z.B. WS-AtomicTransaction und WS-BusinessActivity darstellt. „Ein WS-Coordination-Service besteht aus einem Activation Service (zum Erstellen einer neuen Transaktion), einem Registration Service (zum Beitreten zu einer bereits laufenden Transaktion) und mehreren Protocol Services (welche Implementierungen von *coordination protocols* sind)." (Dostal et al. 2005, 245) Die Details eines *protocol services* werden von den *coordination types*, wie z.B. WS-AtomicTransaction, übernommen (vgl. Abbildung 6.3).

6.2.3 WS-AtomicTransaction

WS-AtomicTransaction[9] ist ein *coordination type* und besteht aus den drei *coordination protocols* `Completion`, `Volatile2PC` und `Durable2PC`. Jedes dieser Protokolle definiert eine Menge von Zuständen und Zustandsübergängen, bspw. von „active" über „completing" bis hin zu „ended". „Das Entscheidende an WS-AtomicTransaction ist jedoch,

[7] Neben dem WSTF gibt es das *Business Transaction Protocol* (BTP) unter der Verwaltung von OASIS sowie das *Web Services Composite Application Framework* (WS-CAF) unter der Verwaltung von u.a. Sun Microsystems und Oracle

[8] aktuelle Spezifikation (August 2005) unter `ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf`, verifiziert am 10.02.2006, 14:59 MEZ

[9] aktuelle Spezifikation (August 2005) unter `ftp://www6.software.ibm.com/software/developer/library/WS-AtomicTransaction.pdf`, verifiziert am 10.02.2005, 15:01 MEZ

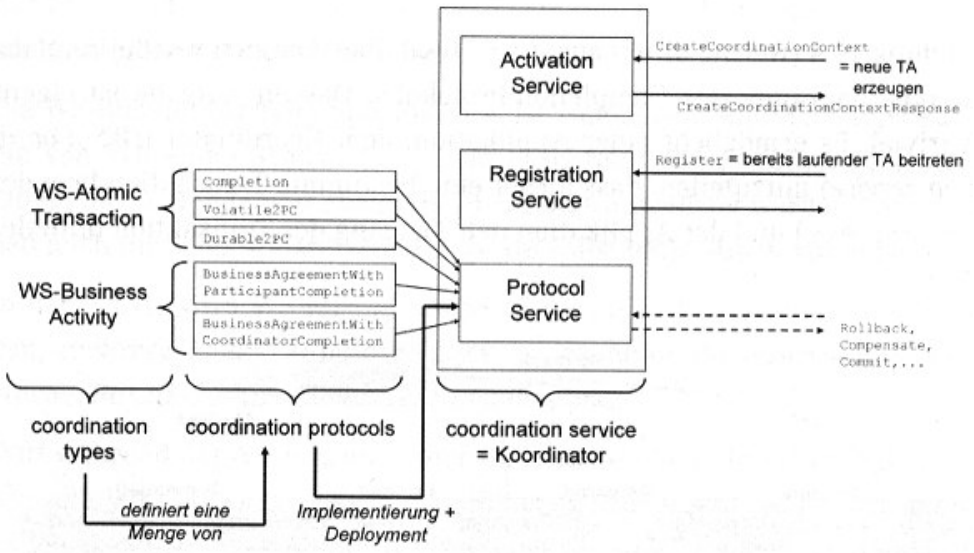

Abbildung 6.3: Überblick über WS-Coordination (Dostal et al. 2005, 245)

dass die Spezifikation (entgegen ihres Namens) in keinster Weise Atomarität fordert! WS-AtomicTransaction kann sogar problemlos dazu verwendet werden, eine Transaktion zu implementieren, welche keine einzige der vier ACID-Eigenschaften erfüllt." (Dostal et al. 2005, 246f.) Jedoch ist in Zukunft zu erwarten, dass WS-AtomicTransactions überwiegend dazu verwendet werden, um kurz laufende und synchrone Transaktionen zu implementieren. Kombiniert man diese (atomaren) Einzeloperationen zu größeren Transaktionen, erhält man eine *BusinessActivity*.

6.2.4 WS-BusinessActivity

Besonders lang andauernde, asynchrone Geschäftsprozesse sind aus mehreren atomaren Operationen zusammengesetzt (hier aus WS-AtomicTransactions). Können nicht alle Teilaufgaben einer WS-BusinessActivity erfolgreich ausgeführt werden, müssen bereits erfüllte Teilaufgaben rückgängig gemacht werden. Dieser Vorgang wird als *Kompensation* bezeichnet.

Zu diesem Zweck wurde eine eigene Spezifikation erstellt, die WS-BusinessActivity[10]. Eine WS-BusinessActivity darf selbst keine Ressourcen exklusiv sperren, sondern lediglich WS-AtomicTransactions als Teil einer größeren Aktivität dürfen dies. Ebenso wie bei WS-AtomicTransaction werden in der Spezifikation bei WS-BusinessActivity Zustände und Zustandsübergänge definiert. Besonders hervorzuheben ist hierbei, dass sogar nach erfolgreichem „Completed" eine Nachricht empfangen werden kann, die die BusinessActivity zum Rückgängigmachen auffordert („Compensate"-Nachricht). Für diesen Fall muss

[10] aktuelle Spezifikation (August 2005) unter `ftp://www6.software.ibm.com/software/developer/library/WS-BusinessActivity.pdf`, verifiziert am 10.02.2006 15:02 MEZ

Code durch den Programmierer erstellt werden, was z.B. im Rahmen der WS-BPEL Spezifikation mittels der „compensation handler“ implementiert wird.

6.3 Zusammenfassung

In diesem Kapitel wurden die Themen Sicherheit und Transaktionen thematisiert, die für den geschäftskritischen Einsatz von Web Services grundlegend sind. Die Aspekte Sicherheit und Transaktionen sind nicht Bestandteil der Basisspezifikationen SOAP und WSDL und werden daher in eigenständigen Spezifikationen gepflegt.

Der Bereich Sicherheit wird durch die Spezifikationen XML Digital Signatures und XML-Encryption abgedeckt. Dazu wird auf bereits etablierte XML-Techniken zurückgegriffen, deren Einsatz aus Gründen der Interoperabilität wiederum durch eine weitere Spezifikation (WS-Security) geregelt wird. Transaktionen treten häufig bei zusammengesetzten Geschäftsprozessen auf. Mittels der Spezifikationen WS-Coordination, WS-AtomicTransaction und WS-BusinessActivity können Transaktionsfunktionalitäten in verschiedener Granularität (atomar für eine Operation oder als BusinessActivity für einen gesamten Geschäftsprozess) bereitgestellt werden.

Die Nutzung von Sicherheits- oder Transaktionsfunktionalitäten ist an das Vorhandensein von Tools geknüpft, die die verschiedenen Spezifikationen implementieren und sich z.B. in vorhandene Web Service Frameworks einfach einbinden lassen. Gerade das Fehlen von nicht-kommerziellen Open-Source-Implementierungen stellt einen zentralen Hinderungsgrund für die Durchsetzung und Verbreitung dieser Techniken dar. Für das Web Service Framework Apache AXIS 1.x existieren prototypische Implementierungen für WS-Security (Apache WSS4J[11]) und WS-Coordination incl. WS-AtomicTransaction und WS-BusinessActivity (im Apache WS-Projekt Kandula[12], jedoch wird im aktuellen Projekt-Status nur der CoordinationType AtomicTransaction unterstützt; BusinessActivies sollen in unbestimmter Zukunft folgen).

Ein weiteres Problem stellt die noch fehlende Erfahrung mit dem Kompensations-Konzept dar, das z.B. im Rahmen von WS-BusinessActivities und WS-BPEL verwendet wird (vgl. Kapitel 5.2.3). „Die meisten Programmierer sind aber noch nicht gewohnt, zu jeder Methode noch eine weitere Methode zu schreiben, die die Effekte der eigentlichen Methode vollständig rückgängig macht.“ (Dostal et al. 2005, 250) Für geschäftskritische Web Services ist dies zur automatischen Fehlerbehandlung eine Voraussetzung.

Mit der Verfügbarkeit von (Open-Source) Tools, die die Spezifikationen zu den Themen Sicherheit und Transaktionen unterstützen, ist mit einem weiter verbreiteten Einsatz von Web Services – auch in geschäftskritischen Bereichen – zu rechnen.

[11] `http://ws.apache.org/wss4j/`, verifiziert am 19.02.2006, 18:47 MEZ
[12] `http://ws.apache.org/kandula/`, verifiziert am 19.02.2006, 18:49 MEZ

7 Web Service Frameworks

Zur Entwicklung und Nutzung von Web Services muss ein Entwickler heutzutage kaum mehr selbst SOAP-Nachrichten erstellen. Leistungsfähige Werkzeuge stehen Entwicklern bei ihrer Arbeit zur Verfügung, so dass sie sich bei der Nutzung von Web Services kaum direkt mit Interna von SOAP-Nachrichten oder WSDL-Dokumenten beschäftigen müssen. Der Umgang mit Web Services wurde dahingehend durch Werkzeuge vereinfacht, dass nicht für jedes Projekt ein eigenes Framework zur Nutzung erstellt werden muss. Dies beschleunigt die Entwicklung und ermöglicht (theoretisch) eine größere Interoperabilität zwischen den verschiedenen Implementierungen, damit ein auf Framework A basierender Web Service ohne Probleme mit einem auf Framework B basierenden Web Service kommunizieren kann.

Die Werkzeugunterstützung ist in der Java- und .NET-Welt mit C# von Microsoft am fortgeschrittensten. Microsoft setzt dabei auf das Visual Studio .Net, während Sun die Java-basierten APIs zur Programmierung von Web Services standardisiert. Diese APIs werden entweder in Form von Referenzimplementierungen (z.B. als Teil von Sun Microsystems „Web Service Developer Pack“ WSDP) oder durch herstellerspezifische Implementierungen und Erweiterungen umgesetzt[1]. Viele – auch kommerzielle – Produkte basieren dabei auf dem AXIS Framework (Apache eXtensible Interaction System), das ein Java-Open-Source Projekt der Apache Software Foundation ist (vgl. Eberhart und Fischer 2004, 208). Auf Grund der freien Verfügbarkeit sowie der weiten Verbreitung dieses Produkts wird in den folgenden Kapiteln dieses Framework schwerpunktmäßig betrachtet.

7.1 Dynamic Invocation und Code-Generierung

Zur Nutzung von Web Services mittels Frameworks gibt es zwei Strategien: Das Generieren von Code (z.B. Java-Code) oder der dynamische Aufruf von Web Services zur Laufzeit (engl. *dynamic invocation*). Beide Strategien werden nachfolgend kurz vorgestellt.

7.1.1 Code-Generierung

Aus einem WSDL-Dokument können mittels Tools beispielsweise die clientseitigen **Stubs** (engl. für Stummel, Stumpf) erzeugt werden, die als Proxy-Klassen fungieren und mit denen über ein Framework ein Web Service angesprochen werden kann (in Abbildung

[1] Beispielsweise von IBM (Rational Rose), Systinet (WASP), IONA (XML-Bus), BEA (WebLogic), Oracle (Oracle 10g), Macromedia (Coldfusion-MX).

7.1 ist dieser Schritt mit 2 gekennzeichnet). Genauso können aus einem WSDL-Dokument die serverseitigen **Skeletons** (engl. für Skelett) erzeugt werden, um ebenfalls wieder über ein Framework einen Web Service anzubieten. In beiden Fällen wird hierbei von einem WSDL-Dokument ausgegangen, von dem aus automatisch Code generiert wird. Dieser Ansatz wird mit „WSDL first“ bezeichnet. Im Gegensatz dazu steht der Ansatz „code first“, bei dem wiederum mittels Werkzeugen z.B. aus vorhandenen Java-Klassen eine Dienstbeschreibung in Form eines WSDL-Dokuments generiert wird.

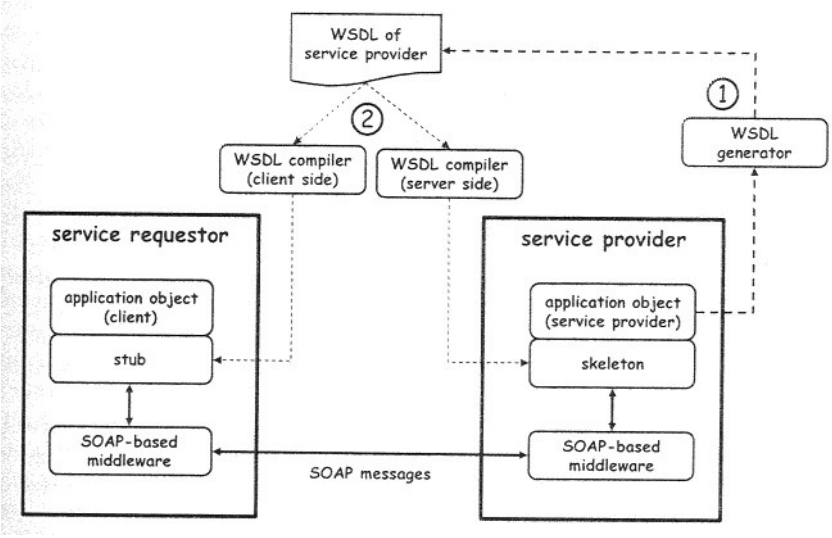

Abbildung 7.1: Einsatz von Werkzeugen zur Generierung von Stubs und Skeletons (Alonso et al. 2004, 173)

Die Nutzung von Stubs und Skeletons ist auf die RPC-Methodologie zurückzuführen (vgl. Dostal et al. 2005, 35ff.). Der scheinbar lokale Aufruf einer Methode wird über den Stub für die Anwendung transparent an den entfernen Zielrechner zur Verarbeitung weitergereicht. „The stub then takes care of locating the server (i.e. *binding* the call to a server), formatting the data appropriately (which involves *marshalling* and *serializing* data), communicating with the server, getting a response and forwarding that response as the return parameter of the procedure invoked by the client.“ (Alonso et al. 2004, 37) Serverseitig findet man ein *Skeleton* als Entsprechung, das den Aufruf des Client-Stubs entgegennimmt, die Daten formatiert (d.h. deserialisiert), die gewünschte Operation mit den Daten ausführt und das Ergebnis wieder zum aufrufenden Client-Stub übermittelt.

Stubs zur Nutzung von Web Services werden überwiegend durch die Programmierer während der Codierungs-Phase erstellt, damit sie von der Applikation zum Compile-Zeitpunkt verwendet werden können. Bei diesem Vorgehen muss also bekannt sein, welche Web Services in einer Applikation verwendet werden sollen, damit entsprechende Stubs (und Hilfsklassen) generiert werden können. Ein flexibles und dynamisches Auswählen von Web Services zur Laufzeit einer Anwendung ist somit nicht möglich.

7.1.2 Dynamic Invocation

Im Gegensatz zur Code-Generierung steht die *Dynamic Invocation*. Dabei werden im Vorfeld keine Stubs generiert, sondern zur Laufzeit können Web Services gebunden und genutzt werden. Dies ist der Hauptvorteil und ein Schritt mehr in Richtung SOA, da auf diese Weise z.B. mittels UDDI ein Dienst gesucht werden und ohne vorherige Code-Generierung dynamisch in einer bestehenden Anwendung eingesetzt werden kann.

Buhler et al. (2004, 3) beschreiben den aktuellen Status der Frameworks hinsichtlich der Unterstützung von *Dynamic Invocation* wie folgt: „Unfortunately, seamless dynamic invocation is beyond the capability provided by these toolkits for the simple reason that they are incapable of handling complex types returned from the invoked service. This limitation is due to the fact that the returned data must be unmarshalled from the SOAP message, which in Java is not possible without having a compatible class that implements the serializable interface."

Zwar bieten einzelne Framworks Möglichkeiten, dynamisch Web Services einzubinden, jedoch unterscheiden sich die dabei eingesetzten Vorgehensweisen grundlegend voneinander. Aus diesem Grund entwickelten Buhler et al. (2004, 9) ein „Composite Pattern for Web Service Invocation (CPWSI)", das unter einem einheitlichen Vorgehen den dynamischen Aufruf von Web Services über unterschiedliche Frameworks ermöglichen kann.

7.2 Ablauf der Verarbeitung mittels Apache AXIS

Bei AXIS muss zwischen **serverseitiger** (Dienstbereitstellung) und **clientseitiger** Verarbeitung (Dienstnutzung) unterschieden werden. Serverseitig ist AXIS als Web-Applikation konzipiert, die prinzipiell in jeden Java-fähigen Web-Server integriert werden kann (z.B. in Apache Tomcat[2]). Das AXIS-Servlet, auch als AXIS-Engine bezeichnet, erhält die vom Applikationsserver weitergeleiteten SOAP-Nachrichten und sorgt für die Verarbeitung der Anfrage, was folgende Schritte umfasst (Eberhart und Fischer 2004, 209):

1. „Parsen der SOAP-Anfrage
2. Lokalisieren der den Service implementierenden Klasse
3. Gegebenenfalls Kompilieren der Klasse
4. Aufruf von so genannten Handlern zur Vorbereitung der Anfrage
5. Aufruf der eigentlichen Implementation
6. Aufruf der Handler zur Nachbearbeitung der Anfrage
7. Kodierung der Ergebnisse"

[2] `http://tomcat.apache.org/`, verifiziert am 10.02.2006, 11:08 MEZ

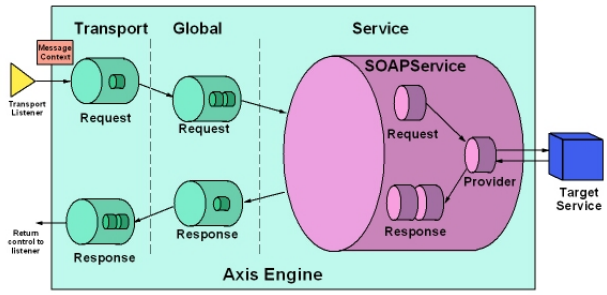

Abbildung 7.2: Ablauf der Verarbeitung innerhalb des AXIS Servers (AxisDoc 2005)

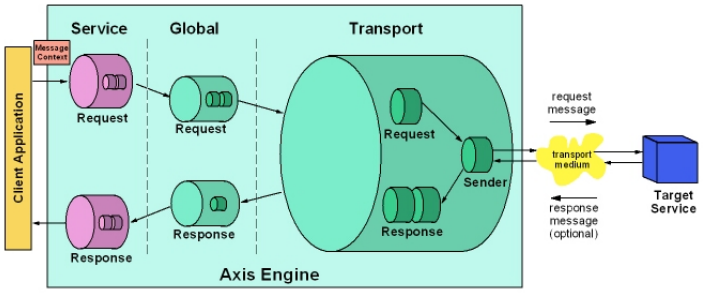

Abbildung 7.3: Ablauf der Verarbeitung innerhalb des AXIS Clients (AxisDoc 2005)

Die AXIS-Engine besteht aus drei Schichten (Transport-, globale Schicht und Serviceschicht) für die jeweils ein oder mehrere *Handler* zur Verarbeitung einer Nachricht vorhanden sind (siehe Abbildungen 7.2 und 7.3). Es gibt z.B. einen Handler in der Transportschicht, der alle via HTTP versandten Nachrichten verarbeitet, während sich ein weiterer Handler um SMTP-Nachrichten kümmert. Welcher Handler eingesetzt wird und die Reihenfolge der Verarbeitung kann in einer Konfigurationsdatei von AXIS festgelegt werden. Handler können zur Kapselung in einer Kette (engl. *chain*) gruppiert werden. Den einzelnen Handlern wird jeweils eine Instanz eines *Message Context*-Objekts übergeben, das alle in der Nachricht enthaltenen Informationen der jeweiligen Anfrage und die dazugehörigen Antworten speichert. (vgl. AxisDoc 2005)

Burghardt (2004, 100ff.) erläutert in allgemeiner Weise das bei AXIS eingesetzte Handler-Konzept. Dieses ermöglicht das Hinzufügen von Erweiterungen bei der Verarbeitung von SOAP-Nachrichten. Beispielsweise können auf diese Weise Funktionalitäten zur Verschlüsselung oder Authentifizierung ergänzt werden (z.B. spezifiziert durch XML-Encryption und XML Digital Signature, siehe Kapitel 6.1.1 oder WS-Security, siehe Kapitel 6.1.2), um letztlich inkrementell die Protokolle im Web Service Stack mit konkreten Implementierungen zu nutzen.

7.2.1 AXIS (Version 1.3)

Zu Beginn gab es Apache SOAP, eine erste „proof of concept"-Implementierung eines Web Service Frameworks, welches im Rahmen eines IBM Research Projekts entstanden ist und anschließend der Open-Source-Community zugänglich gemacht wurde. Daher wird der Nachfolger Apache AXIS 1.x[3] als (SOAP-)Framework oder Web Service Middleware der zweiten Generation beschrieben, wobei die Weiterentwicklung von AXIS 1.x zu AXIS2 als dritte Generation bezeichnet wird (vgl. Perera 2005).

AXIS 1.x unterstützt SOAP 1.1 und 1.2, WSDL 1.1 und folgt dem WS-I Basic Profile 1.0, um eine interoperable Nutzung von Web Services verschiedener Frameworks zu garantieren.

[3] `http://ws.apache.org/axis`, verifiziert am 04.02.2006, 16:05 MEZ

Neben der Client- und Serveranwendung werden mit AXIS sog. „Generatoren“ mitgeliefert (vgl. Kapitel 7.1.1):

- **WSDL2Java:** Dieses Tool erzeugt aus einem WSDL-Dokument die client- und serverseitigen Stubs und Skeletons.
- **Java2WSDL:** Ausgehend von einer bestehenden Web Service Implementation erzeugt dieses Tool automatisch aus den Java-Klassen ein WSDL-Dokument. Dieses Tool kommt vorwiegend dann zum Einsatz, wenn ein (AXIS-)Web Service durch Anhängen von „?WSDL“ an die Service-Endpunkt-URL aufgerufen wird, um die WSDL-Beschreibung übermittelt zu bekommen. „Jedoch ist dies im Hinblick auf die Interoperabilität mit Vorsicht zu genießen. So ist zum Beispiel bei Axis [1.x] der Dokumenten Typ standardmäßig auf RPC/Encoded gesetzt. Dieser muss für den Fall der Interoperabilität auf Document/Literal oder sogar auf Wrapped gesetzt werden.“ (Bierkoch 2005, 38f.) (Vgl. hierzu Kapitel 4.1.2 und Kapitel 4.2.2.)

7.2.2 AXIS2 (Version 0.91)

Apache AXIS2[4] ist eine grundlegende Neuentwicklung eines Open-Source Web Service Frameworks auf den bislang mit AXIS 1.x gemachten Erfahrungen und den Fortschritten im Bereich von Web Services. Besonders wurde hierbei auf eine bessere Performance und einen geringeren Speicherverbrauch geachtet. Dies ist bei Axis 1.x bei großer Last mit vielen gleichzeitigen Nachrichten problematisch. Änderungen erfolgten insbesondere hinsichtlich (vgl. Perera (2005), Perera und Ranabahu (2005) und Frotscher (2005)):

- **eines neuen SOAP Objekt-Modells:** AXIS 1.x benutzt zur XML-Verarbeitung den SAX-Parser, der, nachdem das Parsen gestartet wurde, nicht mehr angehalten werden kann („push“ parsing). Daher müssen auch große XML-Dateien im Speicher gehalten werden. AXIS2 führt ein neues Objekt-Modell (AXIOM genannt) ein, dass eine „pull“ parsing-Strategie verwendet. D.h., Objekte eines XML-Baumes werden nur erzeugt, wenn sie tatsächlich benötigt werden damit nicht unnötig geparst wird. Dazu wird das StAX-API eingesetzt (Streaming API for XML), das Geschwindigkeits- und Laufzeitvorteile mit sich bringt.
- **einer besseren Unterstützung der Nachrichtenfluss-Steuerung:** Während AXIS 1.x ausschließlich auf ein Request-Response-Schema beim Nachrichtenaustausch setzte (in Anlehnung an einen sofortigen, beidseitigen Nachrichtenaustausch mittels RPC), werden von AXIS2 die „Message Exchange Patterns“ (MEPs) der WSDL 2.0 Spezifikation direkt unterstützt (siehe Kapitel 4.2.3).
- **einer verbesserten Unterstützung von asynchronen Web Service Aufrufen:** Bei verteilten Anwendungen wurde überwiegend auf RPC gesetzt, bei denen eine sofortige Antwort auf eine Anfrage erwartet wurde. Service-oriented Architectures hingegen zeichnen sich durch manchmal längere Antwortzeiten aus (z.B. Minuten

[4] `http://ws.apache.org/axis2`, verifiziert am 04.02.2006, 16:05 MEZ

oder länger), vor allem wenn z.B. eine Station eines modellierten Geschäftsprozesses einen menschlichen Arbeitsschritt und dessen Bestätigung abwarten muss. Daher muss dem Nutzer eines Web Services die Möglichkeit gegeben werden, ohne Abwarten einer Antwort und ohne zu blockieren mit seiner Verarbeitung fortzufahren. AXIS2 soll dies mit seiner neuen API ermöglichen. „Die Klasse *Call* bietet nun Methoden, mit denen man bestimmen kann: Sende eine Nachricht mit Transportprotokoll A und empfange die Antwort mit Transportprotokoll B." (Frotscher 2005) Beispielsweise kann eine Anfrage via HTTP versandt, die Antwort aber asynchron via SMTP erwartet werden. Dieses Szenario ist mit AXIS 1.x nicht umsetzbar.

- **einer verbesserten Handler-Unterstützung:** Bei AXIS 1.x konnte die Reihenfolge der Handler nur durch den Entwickler vor dem Deployment festgelegt werden. Sollen weitere Handler z.B. für WS-Security integriert werden, muss auf deren erforderliche Verarbeitungsreihenfolge Rücksicht genommen werden. AXIS2 erleichtert dies durch Einführung des Konzeptes einer Phase (engl. *phase*) und steuernden *phase rules*. Somit können (WS-)Erweiterungen modular hinzugefügt werden.
- **einer modular gestalteten Möglichkeit zum XML Data Binding:** Innerhalb von AXIS2 können verschiedene XML Data Binding Verfahren (z.B. XMLBeans, JAXB oder Castor) verwendet werden, je nach Präferenz und Erfordernissen des Programmierers.

7.3 Web Services Invocation Framework (Version 2.0.1)

Das *Web Services Invocation Framework* (WSIF) wurde ursprünglich von IBM auf alphaWorks veröffentlicht, wurde aber inzwischen zu einem Apache Open-Source Projekt[5]. WSIF ist eine Java API, die es ermöglicht, Web Services unabhängig vom verwendeten Transportprotokoll anzusprechen, d.h. auch andere Protokolle als SOAP zu verwenden. Fremantle (2002) gibt als Motivation dafür an, dass es (zum Zeitpunkt der Veröffentlichung seines Artikels im Jahre 2002) viele andere Protokolle und Technologien für verteilte Anwendungen gibt, die besonders hinsichtlich Transaktionsunterstützung, Sicherheit und Quality of Service mehr zu bieten haben als SOAP. Zudem hätten Firmen bereits in andere Technologien wie z.B. CORBA investiert und wollten diese Infrastruktur weiter nutzen. „With WSIF, the user programs to the abstract service representations, instead of programming to a specific client-side implementation of a protocol such as SOAP." (Duftler et al. 2001, 6)

„WSIF analysiert die WSDL und überträgt die konkreten Aufgaben an sog. Provider. Ein Provider behandelt dann protokollspezifisch die gestellten Anforderungen an die Kommunikation mit den konkreten Diensten." (Rost 2003). Als besonderer Vorteil wird die Unabhängigkeit der WSIF-API von der konkreten Implementierung eines Providers betont, z.B. falls der SOAP Provider ausgetauscht wird (statt Apache SOAP nun Apache

[5] `http://ws.apache.org/wsif`, verifiziert am 05.02.2006 13:30 MEZ

AXIS). Dies zwingt nicht zu einer Anpassung des Codes, wie er beim Wechsel eines Web Service Frameworks evtl. vonnöten gewesen wäre. In der aktuellen Version werden Provider zur Verfügung gestellt für Apache SOAP und AXIS (in Version 1.3), lokale Java-Objekte, Enterprise Java Beans (EJB), Java Messaging Services (JMS) und J2EE Connector Architecture (JCA).

WSIF unterstützt zwei Arten von Web Service-Aufrufen: Die Nutzung von generierten Stubs (via AXIS) oder das Prinzip der Dynamic Invocation (vgl. Kapitel 7.1). In den FAQs zur aktuellen WSIF Version erfolgt eine Einschränkung: „WSIF does support complex types – of course XML schema support is limited (but very reasonable). The DynamicInvoker [class] doesn't support invocation of services using complex types since this requires that java representations of the complex types be generated first."

7.4 Zusammenfassung

Dieses Kapitel beschäftigte sich mit Web Service Frameworks. Alle Frameworks unterstützen die clientseitige Nutzung von Web Services entweder durch Generierung von Proxy-Klassen (Stubs) oder durch Nutzung von Dynamic Invocation. Dem reizvollen Konzept der Dynamic Invocation stehen im praktischen Einsatz einige Probleme gegenüber. Allen voran ist dies die mangelnde Unterstützung von komplexen Datentypen, was besonders im praktischen Teil der Arbeit von besonderer Relevanz war (vgl. Kapitel 8.4.2).

Die Open-Source Frameworks AXIS (1.x und 2) und WSIF wurden in diesem Kapitel mit ihren Grundkonzepten vorgestellt. Eine Bewertung hinsichtlich ihrer Fähigkeiten, Stärken und Schwächen erfolgt in Kapitel 8.2, um die Entscheidung für ein bestimmtes Framework im Rahmen der Implementierung des praktischen Teils dieser Arbeit zu begründen. Das folgende Kapitel beschreibt die Herangehensweise zur Umsetzung des praktischen Teils der Masterarbeit.

8 Implementierung des Prototyps

Die folgenden Kapitel beschreiben den praktischen Teil dieser Arbeit: Die Erstellung eines prototypischen „Web Service Import Wizards“ für den abaXX Process Modeler. Dieser Prototyp soll den Einsatz von Web Services als Aktivitäten innerhalb eines Prozessmodells ermöglichen. Die Dienste, die ein Web Service bereitstellt, werden nach dem „Import-Vorgang“ als Aktivitäten innerhalb der Activities Gallery des Process Modelers angezeigt, um sie mittels Drag&Drop zur Gestaltung eines Prozesses einzusetzen (siehe Kapitel 2.5).

Ausgehend von Anwendungsfällen werden im Anschluss die getroffenen Entscheidungen für eine Implementierung erläutert. Darauf folgen eine Beschreibung des Vorgehens während der Implementierung, eine kurze Vorstellung des Resultats und eine Diskussion der Problembereiche und Besonderheiten, auf die im Rahmen dieses praktischen Teils gestoßen wurde.

8.1 Anwendungsfälle

Zum graphischen Modellieren von Prozessen bietet die abaXX-Lösung den sog. *„Process Modeler“* an. Mit diesem Werkzeug können Prozesse mittels Drag&Drop erstellt werden. Sie werden anschließend als Prozessdefinition in Form einer XML-Datei gespeichert (vgl. Kapitel 2.4). Das ist der Anknüpfungs- und Ausgangspunkt für den praktischen Teil dieser Arbeit.

Aus dem in Kapitel 2.5 (Seite 10) vorgestellten Nutzungsszenario und den darin formulierten Anforderungen an einen prototypischen „Web Service Import Wizard“ werden im nächsten Schritt einzelne Anwendungsfälle (engl. *use cases*) extrahiert. Sie dienen in der Fortführung der Analysephase dazu, die Grundlagen für die Entwurfs- und Implementierungsphase zu legen. Folgende Anwendungsfälle werden anschließend detailliert beschrieben:

- Starten des „Web Service Import Wizards“ zur Parameter-Eingabe
- Extrahieren von Informationen aus einem WSDL-Dokument
- Erstellen von Activities ausgehend von einem WSDL-Dokument
- Erstellen der clientseitigen Stubs

8.1.1 Starten des „Web Service Import Wizards“ zur Parameter-Eingabe

Beschreibung: Der Nutzer möchte im Rahmen der Prozessmodellierung mittels des Process Modelers auf bereitgestellte Dienste von Web Services zurückgreifen. Diese sollen als Aktivitäten in der Activities Gallery dargestellt werden und können daraus in das Prozessmodell per Drag&Drop eingebunden werden. Durch Angabe einer WSDL-URL innerhalb des „Web Service Import Wizards“ werden die vom Web Service angebotenen Dienste ausgelesen und als Aktivitäten dargestellt.

Beteiligte Akteure: Anwender, Wizard

Auslöser: Die Aktion wird vom Anwender explizit ausgelöst.
(Starten des „Web Service Import Wizards“)

Vorbedingungen:

1. Process Modeler ist geöffnet.
2. Die URL einer WSDL muss dem Nutzer bekannt sein.

Ablaufschritte:

1. Nutzer öffnet den „Web Service Import Wizard“.
2. Eingabe der URL einer WSDL-Datei durch den Nutzer
3. Extrahieren von Informationen aus einem WSDL-Dokument
4. Auswahl durch den Nutzer des:
   - ❏ Services (bei mehreren Services)
   - ❏ Binding (SOAP, JMS, EJB . . .)
   - ❏ Port Type
5. Erstellen von Activities ausgehend von einem WSDL-Dokument
6. Nachricht an den Nutzer bei Erfolg oder Fehler

8.1.2 Extrahieren von Informationen aus einem WSDL-Dokument

Beschreibung: Ausgehend von einer WSDL-URL erstellt der „Web Service Import Wizard“ eine Übersicht über die bereitgestellten Dienste eines Web Services. Der Nutzer kann aus dieser Übersicht auswählen, welche Services mit welchem Binding über welchen Port Type (mit allen vorhandenen Operationen) als Aktivitäten in der Activities Gallery des Process Modelers bereitgestellt werden sollen.

Beteiligte Akteure: Anwender, Wizard

Auslöser: Nutzer gibt eine URL einer WSDL ein und drückt auf Button „Next“ im „Web Service Import Wizard“

Vorbedingungen:

1. Process Modeler ist geöffnet.
2. URL einer WSDL-Datei liegt vor.
3. URL wurde auf Gültigkeit geprüft.
4. Vorhandensein einer WSDL-Datei wurde überprüft.

Eingesetzte Tools: WSDL4J

Ablaufschritte:

1. Auslesen der vorhandenen Services (falls mehrere Services in einer WSDL integriert sind)
2. Auslesen der möglichen Binding-Arten (z.B. SOAP, JMS etc.)
3. Auslesen der möglichen PortTypes
4. Auslesen der möglichen Operationen (= Aktivitäten für die Activities Gallery)
5. Nutzer wählt benötigte Services/Bindings/Port Types aus
6. Nutzer klickt auf Button „Finish“

8.1.3 Erstellen von Activities ausgehend von einem WSDL-Dokument

Beschreibung: Ausgehend von einer WSDL-URL und der Auswahl des Nutzers (Services/Bindings/Port Type) erstellt der „Web Service Import Wizard“ die Aktivitäten, die in der Activities Gallery im Process Modeler dargestellt werden. Zum Ansprechen des Web Services über ein Web Service Framework werden Stub-Klassen benötigt, die ebenfalls während dieses Importvorgangs automatisch erzeugt werden.

Beteiligte Akteure: Wizard

Auslöser: Im „Web Service Import Wizard“ klickt der Nutzer auf „Finish“.

Vorbedingungen: Nutzer startet den „Web Services Import Wizard“ und legt Angaben fest zu Services/Binding/Port Type (Auswahl wurde zuvor auf Plausibilität geprüft).

Eingesetzte Tools: Apache Velocity als Template-Mechanismus zur Erstellung der Datei activities.xml

Ablaufschritte:

1. Erstellen der clientseitigen Stubs
2. Variablen-Mapping zwischen In-/Output des Web Services und der zu erstellenden Aktivität
3. Erstellen der Aktivität als XML-Datei activities.xml für einen späteren Import in die Activities Gallery des Process Modelers

8.1.4 Erstellen der clientseitigen Stubs

Beschreibung: Um auf bereitgestellte Dienste eines Web Services zurückzugreifen, müssen Stubs erstellt werden, die die Kommunikation zwischen Teilen des Prozessmodells (in einer Java-Runtime Umgebung) und einem Web Service ermöglichen. Stubs können entweder manuell erzeugt werden (aufwändig) oder automatisch durch Einsatz von Tools eines Web Service Frameworks, z.B. WSDL2Java von AXIS (entspricht der in diesem Falle gewählten Implementierungsart).

Beteiligte Akteure: Wizard

Auslöser: Nutzer klickte auf „Finish“ im Schritt 2 des „Web Service Import Wizards“

Vorbedingungen: vgl. Vorbedingungen zu Anwendungsfall „Erstellen von Activities ausgehend von einem WSDL-Dokument“

Eingesetzte Tools: Apache AXIS 1.3

Ablaufschritte:

1. Speicherort für Stub-Klassen festlegen (in externer Properties-Datei für den Wizard)
2. Aufrufen des Tools zur Stub-Erzeugung
3. Auswerten des Rückgabewerts des Tools für evtl. Fehlermeldung an den Nutzer

8.2 Auswahl eines Web Service Frameworks

Ein SOAP Framework-Hersteller (XFire) veröffentlicht einen regelmäßig aktualisierten Vergleich zwischen verschiedenen Web Service Frameworks, der hier wiedergegeben wird (siehe Tabelle 8.5, `http://xfire.codehouse.org/Stac+Comparison`, verifiziert am 05.02.2006 14:13 MEZ).

	ActiveSOAP (CVS)	**Axis 1.2.x**	**Axis 2 (0.9)**	**Glue**	**XFire**
Basic Profile 1.1 Compliant	X	X	X	X	X
Easily Embedded	X		X	X	X
DIME		X		X	
Easily Create Services from POJOs				X	X
JAXB 1.1		X			X
JAXB 2.0	X				X
JAX-RPC		X		X?	
JAX-WS					Early Access
JBI		?			X
JSR 181		X			X
JSR 181 via commons-attributes					X
MTOM			X		Scheduled for 1.1
Open Source	X	X	X		X
Soap 1.1	X	X	X	X	X
Soap 1.2	X	X	X	X	X
Soap with Attachments		X	?	X	Scheduled for 1.1
StAX based	X		X		X
WSDL 1.1 Support		X	X	X	X
WSDL 2.0 Support			Post 1.0		Scheduled for 1.1
WSDL -> Code (Client)		X	X	X	X
WSDL -> Code (Server)		X	X	X	X
XMLBeans	X	X	X		X
HTTP	X	X	X	X	X
JMS	X	X		X	X
Jabber		Experimental			X
SMTP		X	X		
WS-Addressing		X	X	X	X
WS-Security		X	X	X	(in CVS)
WS-Notification		?	?	?	Via ServiceMix

Tabelle 8.5: Vergleich von Web Service Frameworks

Aus der Tabelle kann man ablesen, dass die verschiedenen Web Service Frameworks in ihrer Entwicklung und den implementierten Fähigkeiten große Unterschiede aufweisen. Manche, evtl. sogar wichtigen Komponenten, finden sich nur bei dem einen oder bei dem anderen Framework.

Ein weiteres Web Service Framework ist mit dem von Sun Microsystems entwickelten *Web Service Developer Pack* verfügbar. Auf Grund der bei Bierkoch (2005, 41 und 47) sowie Apfel (2004, 50ff.) beschriebenen Nachteile und Einschränkungen dieses Frameworks (z.B. geringere Unterstützung von XML Datentypen im Vergleich zu AXIS 1.x), wurde es im Vorfeld als möglicher Kandidat für die Implementierung im Rahmen dieser Arbeit ausgeschlossen.

Die von AXIS2 aufgeführten Vorteile legen es einem Entwickler nahe, sich für AXIS2 als Web Service Framework zu entscheiden. Da sich dieses Tool noch im Entwicklungsstadium (Version < 1.0) befindet, unterliegen diese beschriebenen Vorteile jedoch einigen

Einschränkung: Die Entwickler von AXIS2 wollen nicht sämtliche Programm-Features in der ersten Release-Version implementieren, sondern, so Frotscher (2005), inkrementell vorgehen. Wichtige Möglichkeiten und Fähigkeiten könnten so womöglich relativ spät implementiert werden. Zudem erschweren die bislang kaum vorhandenen Beispiele und die spärlich vorhandene Dokumentation einen Einstieg und ein praktisches Überprüfen der Leistungsfähigkeit von AXIS2 enorm. Im Rahmen dieser Arbeit wurde versucht, das Development-Release in Version 0.91 anhand der mitgelieferten Beispiele zu testen, was nicht vollständig gelang. Die Nutzung eines eigens entwickelten Beispiel-Web Services mit komplexen Datentypen konnte auf Grund fehlender Dokumentation und unklarer Vorgehensweise aus den Beispielen nicht erfolgreich umgesetzt werden. Aus diesen Gründen ist ein Produktiveinsatz von AXIS2 in einer Entwicklungsversion noch nicht sinnvoll. Die Verwendung von AXIS2 in einer stabilen Release-Version als Framework ist in Zukunft dagegen sicherlich empfehlenswert.

WSIF verfolgte als Projekt gute Ansätze. Jedoch ist in der Entwickler- und Nutzer-Mailingliste seit geraumer Zeit eine geringe Beteiligung festzustellen (im Vergleich zu einer sehr lebendigen AXIS und AXIS2 Mailingliste), was den Eindruck erweckt, dass WSIF keine stringente Weiterentwicklung und Nutzung erfahren würde. Im Rahmen von Web Services wird speziell das SOAP Protokoll seine Vormachtsstellung behaupten können: Auf Grund seiner bislang schon weiten Verbreitung, der vorhandenen Tool-Unterstützung und den bereits implementierten und geplanten Erweiterungen, wie z.B. WS-Security oder WS-Coordination, wird das in Kapitel 7.3 angebrachte Argument von Fremantle bzgl. der Unausgereiftheit und Mängel von SOAP überwiegend entkräftet.

Aus Gründen der Generalität und Einfachheit wäre eine Lösung zu bevorzugen, die die in Kapitel 7.1 vorgestellte Dynamic Invocation unterstützt. Dieser Ansatz wird generell von allen drei beschriebenen Frameworks unterstützt, jedoch in keinster Weise umfassend. Als Testfall wurde die Nutzung eines Web Services gewählt, der als Rückgabewerte ein Array von (komplexen) Objekten zurückgibt.[1] Alle drei Frameworks benötigten zur Deserialisierung der Rückgabeobjekte die im Rahmen der Stub-Generierung erstellten Objektklassen. Diese Objektklassen repräsentieren die Java-Objekte, die als XML Schema Datentypen innerhalb des WSDL-Dokuments definiert sind. Auf Grund dieser Einschränkung muss für den praktischen Teil dieser Arbeit auf die Nutzung der Dynamic Invocation verzichtet werden.

Betrachtet man alle praktisch gemachten Beobachtungen und die Beurteilungen aus Sekundärquellen, kommt man für die Auswahl eines Web Service Frameworks zu folgendem Schluss: Um Web Services zu nutzen, bietet sich momentan besonders das Apache AXIS Framework in Version 1.3 mit Generierung von Stubs und Skeletons an. Dieses Framework ist bislang am besten dokumentiert, läuft stabil und einige Erweiterungen (z.B. WS-Security oder WS-Addressing) sind bereits implementiert.

[1] Konkret wurden Testfälle mit den von Google und Amazon bereitgestellten Web Services durchgeführt, bei denen als Rückgabewerte Arrays mit komplexen Datentypen verwendet werden.

8.3 Implementierung

Vor dem eigentlichen „Import“ von Web Services in Form von Aktivitäten in die Activities Gallery des Process Modelers werden vom „Web Service Import Wizard“ zunächst Informationen gesammelt (z.B. URL des WSDL-Dokuments), die anschließend von der Anwendungslogik verarbeitet werden. Die dazu eingesetzten Werkzeuge und die Detailstruktur des Ablaufs sind Gegenstand dieses Kapitels. Außerdem werden Schwierigkeiten und Herausforderungen aufgezeigt, die durch die Einbindung von Web Services in eine Web Applikation entstehen.

8.3.1 Eingesetzte Werkzeuge

Die **GUI-Komponenten** des abaXX Process Modelers werden auf Basis von Java Swing erstellt. Diese Bibliothek verfügt nicht über vorgefertigte Wizard-Klassen. Dies hätte es erforderlich gemacht, die Architektur und das Zusammenspiel der einzelnen Wizard-Seiten mit einer Wizard-Steuerungslogik vollkommen selbst zu gestalten. Der Hersteller abaXX Technology beabsichtigt in Zukunft, den Process Modeler zusätzlich als Plug-In innerhalb der Entwicklungsplattform Eclipse nutzbar zu machen. Zur Darstellung der GUI-Komponenten von Eclipse wird die SWT-Architektur (Standard Widget Toolkit) verwendet.[2] JFace erweitert die Funktionen von SWT und dient als Schnittstelle zwischen SWT und der Eclipse-Plattform. Da die JFace-Bibliotheken fester Bestandteil von Eclipse sind, sind diese nicht als eigenständige Distribution erhältlich, sondern nur in Verbindung mit Eclipse. Über JFace ist eine Trennung von Modell und Darstellung möglich, ähnlich dem Model-View-Controller (MVC) Pattern bei Swing.

JFace bietet eine bereits vorhandene Unterstützung von Wizard- oder Assistenten-Klassen für Eclipse-Anwendungen. Aus diesem Grund wurde der Prototyp des „Web Service Import Wizard“ unter Nutzung von SWT/JFace-Bibliotheken erstellt und in den Process Modeler auf Swing-Basis integriert. Zu diesem Zweck bietet das SWT-Framework die Möglichkeit, eine in beiden Richtungen funktionierende AWT/Swing-SWT-Bridge einzusetzen, um Elemente eines graphischen Frameworks innerhalb eines Anderen darzustellen und zu nutzen.

Als **Web Service Framework** wird Apache AXIS in der Version 1.3 eingesetzt (zur Begründung vgl. Kapitel 8.2). Darin enthalten ist das Werkzeug *WSDL2Java*, das zum Erstellen von clientseitigen Stubs und – in diesem Falle nicht benötigt – serverseitigen Skeletons anhand eines WSDL-Dokuments eingesetzt wird (vgl. Kapitel 7.1.1). Zum **Auslesen von**

[2] Dieses API wurde im Zuge der Eclipse-Entwicklung als Open Source Software erstellt. Statt auf die Java Foundation Classes (JFC) zuzugreifen, werden z.B. Bäume und Tabellen über native Komponenten des jeweiligen Betriebssystems dargestellt, was bspw. einen schnelleren Aufbau der GUI im Vergleich zu den JFC wie AWT oder Swing ermöglicht (vgl. Spall 2003).

WSDL-Dokumenten wird die Bibliothek WSDL4J[3] verwendet, die eine API zum Zugriff auf WSDL-Elemente beinhaltet.

Zum Erstellen der Activity-Implementierungen wird die Template-Engine Apache Velocity[4] genutzt. Hierbei werden die aus dem WSDL-Dokument extrahierten Informationen an vordefinierte Stellen in eine Vorlage eingefügt, um auf diese Weise eine Java Quellcode-Datei zu erstellen. Die XML-Datei *activities.xml* wird mittels der Bibliothek JDOM erzeugt.

8.3.2 Architektur

Beim „Web Service Import Wizard“ erfolgt eine Trennung in GUI- und Anwendungslogikteile, die sich in der Package-Struktur widerspiegelt. Die GUI-Komponenten sind im Package `wsImport.ui.*` organisiert. Die Anwendungslogik ist im Package `wsImport` untergebracht. Hilfsklassen, die die Anwendungslogik benötigt, werden im Package `wsImport.model` zusammengefasst (vgl. Klassendiagramme in Abbildungen 8.1 und 8.2).

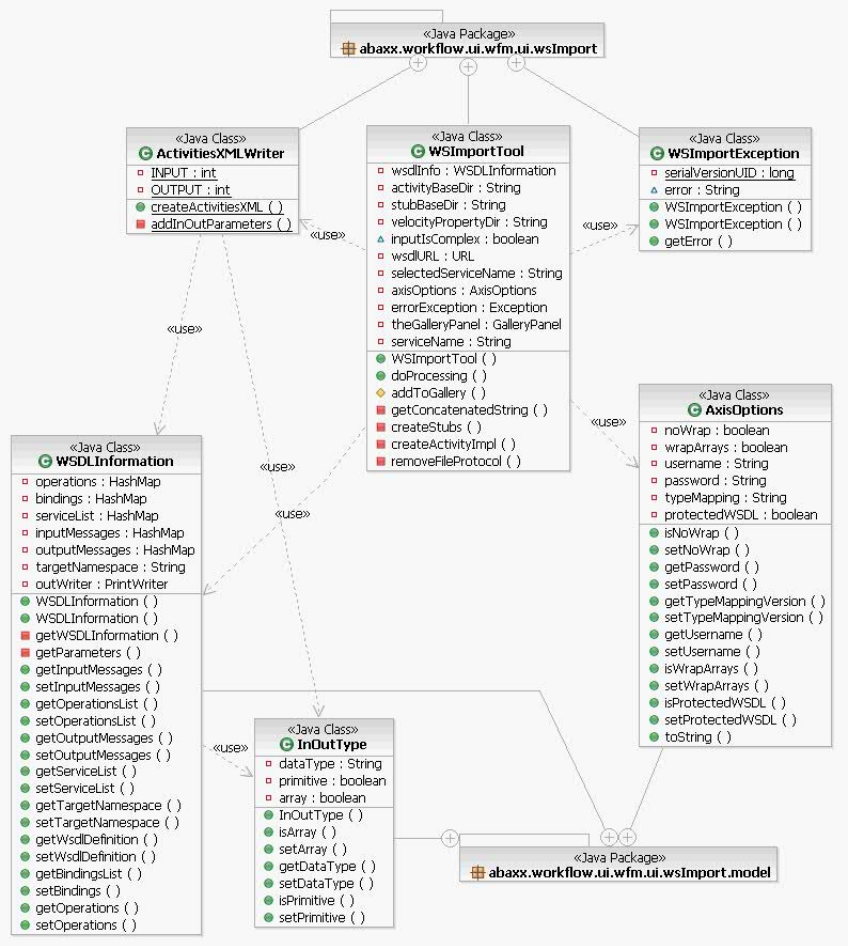

Abbildung 8.1: Klassendiagramm der Packages `wsImport` und `wsImport.model`

[3] WSDL4J ist als Bibliothek der AXIS-Distribution beigefügt.
[4] `http://jakarta.apache.org/velocity/`, verifiziert am 13.02.2006, 11:02 MEZ

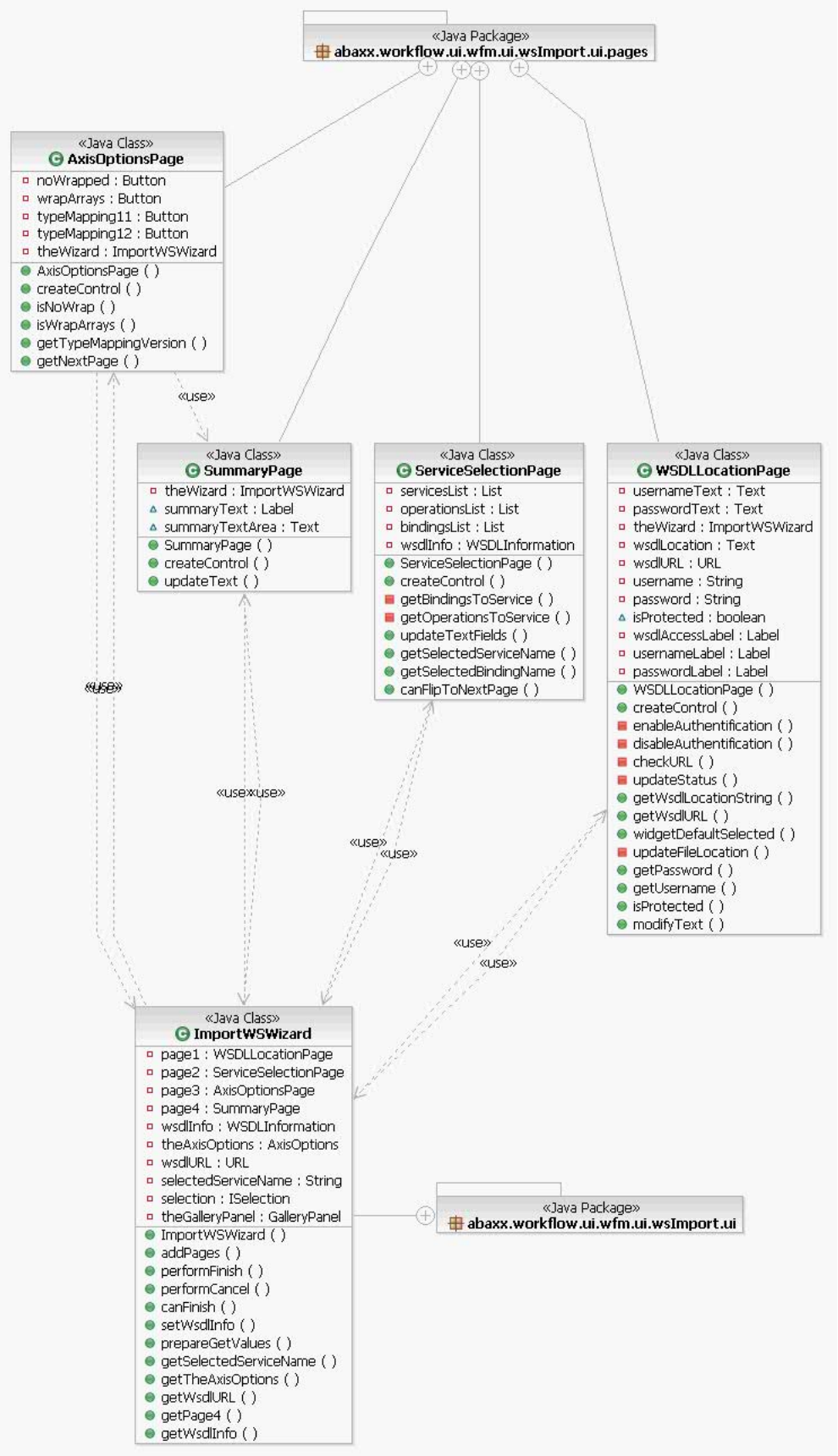

Abbildung 8.2: Klassendiagramm des Packages `wsImport.ui`

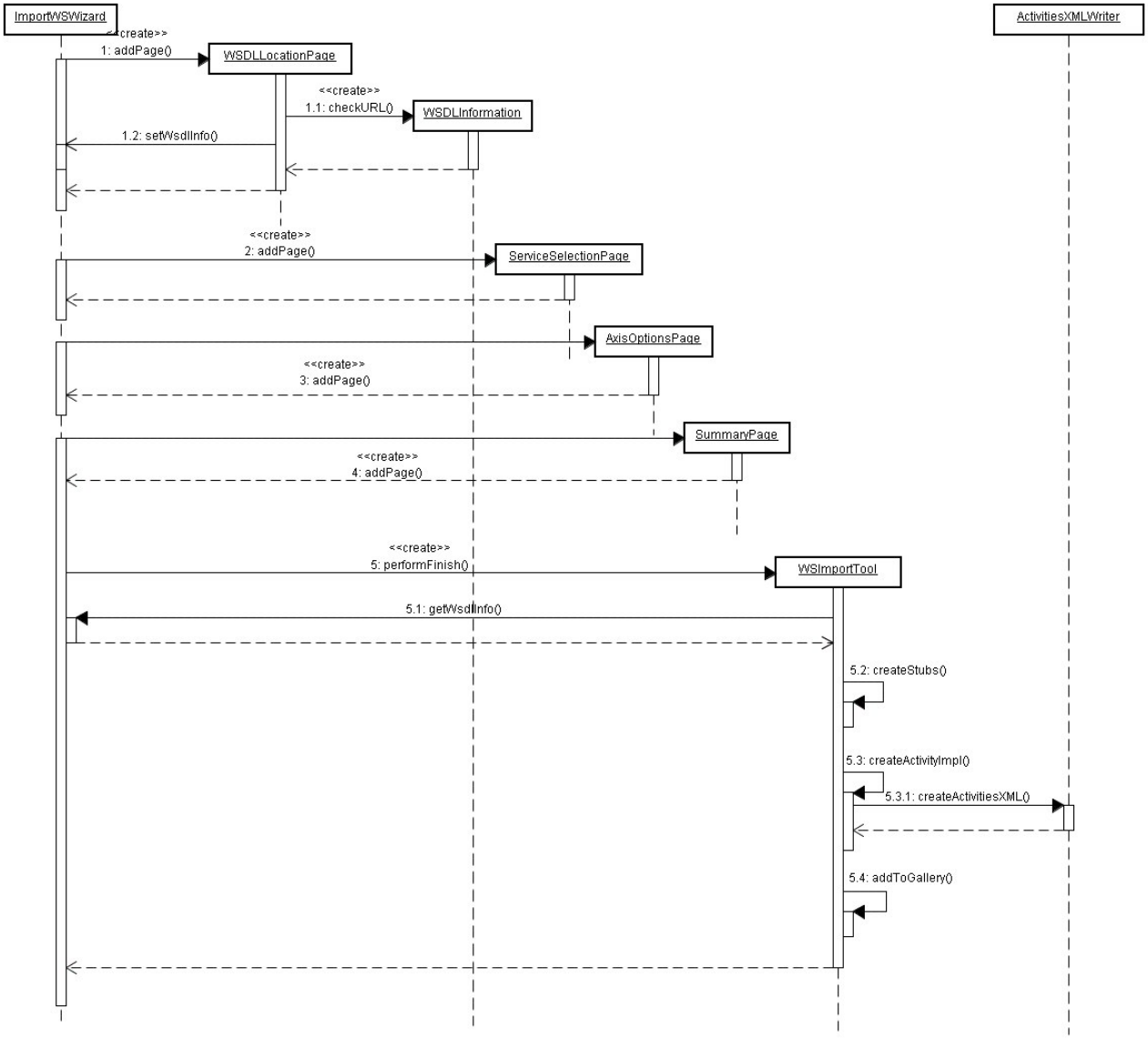

Abbildung 8.3: Sequenzdiagramm des „Web Service Import Wizards“

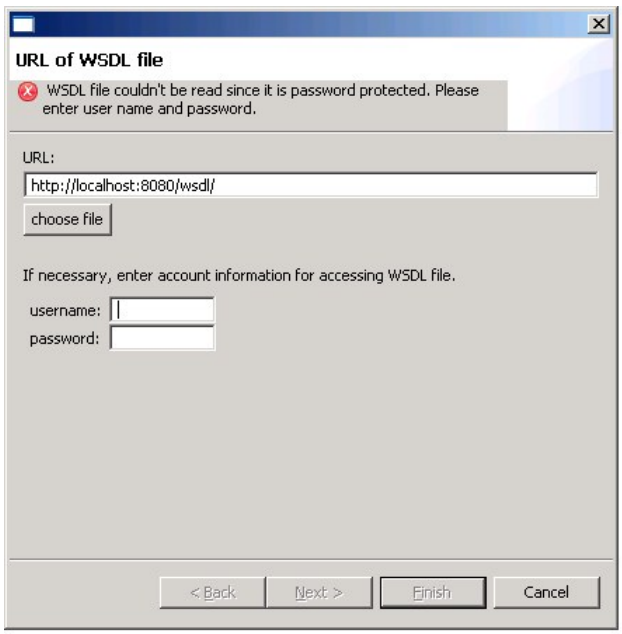

Abbildung 8.4: Eingabe der URL eines WSDL-Dokuments und Fehlermeldung (Schritt 1)

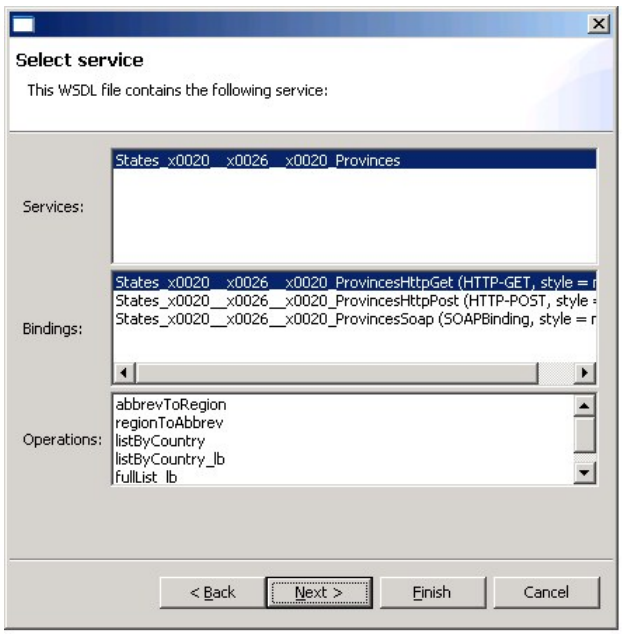

Abbildung 8.5: Auswahl des Services, des PortTypes und des Bindings (Schritt 2)

Die Wizard-Klassen der JFace-Bibliothek sind so organisiert, dass eine Klasse den Ablauf des Wizards steuert (`ImportWSWizard`). Die einzelnen, anzuzeigenden Seiten des Wizards (z.B. `WSDLLocationPage`, `ServiceSelectionPage` etc.) werden in dieser Klasse registriert. Deren Reihenfolge und die Freigabe zur Anzeige auf dem Bildschirm werden durch die Steuerungsklasse koordiniert. Die abschließende Verarbeitung, wenn alle Angaben auf den einzelnen Wizard-Seiten vollständig gemacht wurden, wird ebenfalls durch die Klasse `ImportWSWizard` angestoßen. An dieser Stelle erfolgt die Trennung zwischen GUI- und Anwendungslogik: Sind alle Nutzereingaben komplett durch den Wizard-Dialog erfasst, übernimmt die Anwendungslogik die Erzeugung der Activities und den Import in die Activities Gallery des Process Modelers. Der gesamte Ablauf ist in Form eines Sequenzdiagramms (siehe Abbildung 8.3) illustriert.

8.3.3 Ablauf des „Imports“

Der Nutzer startet den „Web Service Import Wizard“. Im nächsten Schritt wählt er ein WSDL-Dokument aus: Entweder durch Eingabe einer URL oder durch Auswahl über einen „File Dialog“, der über die Schaltfläche „choose file“ zu erreichen ist. Treten Probleme auf, weist eine Fehlermeldung im oberen Teil des Wizard-Dialogs den Nutzer darauf hin (Abbildung 8.4). Folgende Fehler oder Ereignisse werden abgefangen:

- ❏ syntaktische Fehler bei der Eingabe einer URL
- ❏ das WSDL-Dokument enthält keine Daten
- ❏ Fehlende oder fehlerhafte Authentifizierung:
  Das WSDL2Java-Tool von AXIS 1.x erlaubt es, WSDL-Dokumente abzurufen, die mittels einfacher HTTP-Basic-Authentifizierung (z.B. via HTACCESS-Datei eines Apache Webservers) geschützt sind. Der Wizard stellt fest, ob die angegebene URL eine Authentifizierung verlangt und blendet dementsprechend die Eingabefelder für Nutzername und Passwort ein bzw. aus.

Nachdem der Nutzer ein gültiges WSDL-Dokument ausgewählt hat (Schritt 1), wird dieses Dokument zunächst eingelesen und in eine interne Zwischenrepräsentation übersetzt (Klasse `WSDLInformation`). Im zweiten Schritt des Wizard-Dialogs werden die Services, Bindings und Operationen angezeigt, die im zuvor eingelesenen WSDL-Dokument ermittelt wurden (siehe Abbildung 8.5). Der Nutzer hat an dieser Stelle die Möglichkeit, falls mehrere Services in einem WSDL-Dokument angeboten werden, den für ihn Relevanten auszuwählen, für den die Aktivitäten im Process Modeler generiert werden sollen. Zudem besteht die Möglichkeit, falls (wie in Abbildung 8.5 dargestellt) mehrere Bindings existieren, dass der Nutzer eine Auswahl des zu verwendenden Bindings treffen kann. Diese Option zum Auswählen eines Bindings ist für eine zukünftige Erweiterung konzipiert, da das verwendete Web Service Framework SOAP-Nachrichten nur via HTTP, SMTP oder JMS verschicken kann. Andere Bindings werden durch Apache AXIS 1.3 ignoriert.

In Schritt drei des „Web Service Wizards“ (siehe Abbildung 8.6) werden Optionen zur Code-Generierung mittels WSDL2Java abgefragt. Die ausgewählten Optionen werden in einer Hilfsklasse (`AxisOptions`) zwischengespeichert. Im letzten Schritt des Wizards (siehe Abbildung 8.7) werden alle durch den Nutzer getätigten Angaben zu seiner Kontrolle nochmals aufgeführt. Durch Klicken auf die Schaltfläche „Finish“ wird der eigentliche „Import-“Vorgang gestartet.

Als nächstes werden die clientseitigen Stubs unter Berücksichtigung der getroffenen Optionsauswahl (`AxisOptions`) für den Zugriff auf den Web Service erzeugt. Anschließend werden ausgehend von der Zwischenrepräsentation des WSDL-Dokuments (`WSDL-Information`) die Implementierungen der Custom Activities zur Einbindung in die Activities Gallery erstellt. Hierfür wird die Template-Engine Velocity eingesetzt. Für jede in einem WSDL-Dokument definierte Operation wird eine separate Aktivität implementiert. Innerhalb dieser Custom Activity erfolgt der Aufruf der Stub-Klasse zum Ansprechen des Web Services. Die Ein- und Ausgabeparameter einer Operation werden in der generierten Aktivitäten-Implementierung zur Vereinfachung für den Programmierer im Code besonders herausgestellt: Bei Basisdatentypen (z.B. `int` oder `String`) erfolgt pro Operand eine separate Wertzuweisung (siehe Quellcode-Ausschnitt bei den Variablen *aKey* und *aPhrase*, die einen Wert aus dem Prozesskontext zugewiesen bekommen). Bei komplexen Datentypen wird hingegen aus Gründen einer besseren Übersichtlichkeit deren Instantiierung in eine separate Methode ausgelagert (`createObjektName()`).

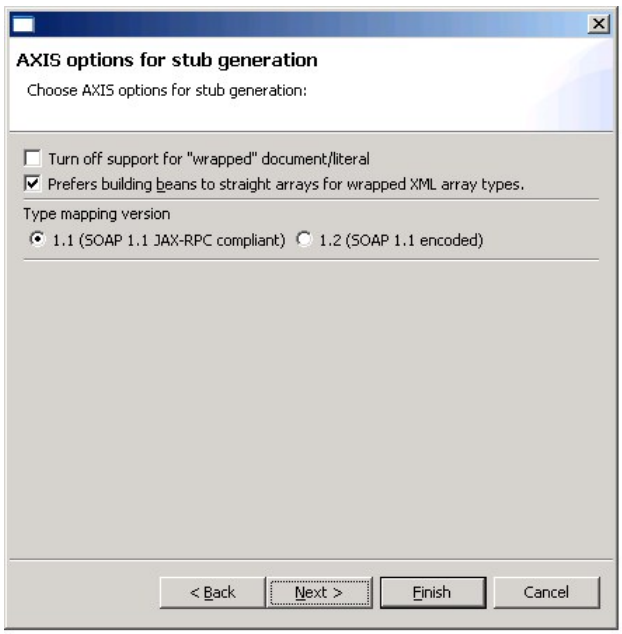

Abbildung 8.6: Auswahl der Optionen zur Code-Generierung (Schritt 3)

Abbildung 8.7: Zusammenfassung anzeigen (Schritt 4)

Erzeugtes Quellcode-Fragment bei einfachen Datentypen:

```
[...] // call WS via stub
java.lang.String response = null;
try {
  // TODO adjust input values
  java.lang.String aKey = (String) context.get("key");
  java.lang.String aPhrase = (String) context.get("phrase");

  response = service.doSpellingSuggestion( aKey, aPhrase );

} catch (RemoteException e) {
  System.err.println("Failed to connect to WS!");
  e.printStackTrace();
  throw new WorkflowException( e.getMessage() );
} [...]
```

Erzeugtes Quellcode-Fragment bei komplexen Objekten:

```
[...] // create ItemSearch object
ItemSearch myRequestObject = createItemSearch( context );

// call WS via stub
ItemSearchResponse response = null;
try
{
  System.out.println("Addressing query to AWSECommerceService...");
  response = service.itemSearch(myRequestObject);
}

[...]
```

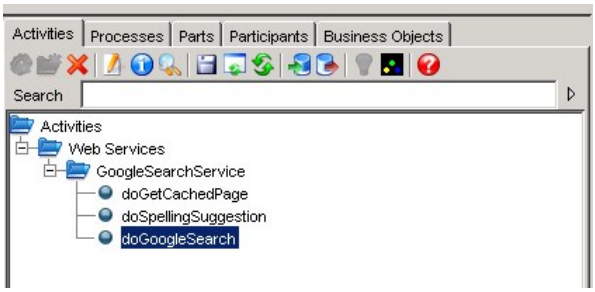

Abbildung 8.8: Ergebnis des „Importierens“ im Process Modeler

```
protected ItemSearch createItemSearch( ProcessContext context )
{
  // create ItemSearch object for sending request to AWSECommerceService
  ItemSearch myRequest = new ItemSearch();

  // TODO: add your code here!
  return myRequest;
 } [...]
```

Nachdem die Aktivitäten-Implementierungen erzeugt worden sind, müssen sie in die Activities Gallery des Process Modelers übernommen werden. Hierzu wird der bestehende Import-Mechanismus des Process Modelers verwendet, der auf dem Einlesen eines XML-Dokuments basiert. Dieses XML-Dokument (activites.xml) beinhaltet eine Beschreibung der Aktivitäten mit ihren Ein- und Ausgabeparametern sowie weitere Angaben. Aus der Zwischenrepräsentation des WSDL-Dokuments wird die Datei activities.xml erstellt und der Importvorgang gestartet. Innerhalb der Activities Gallery wird in der Untergruppe „Web Services“ eine weitere Untergruppe mit dem Namen des importierten Web Services erzeugt. Die Aktivitäten zum Einbinden in ein Prozessmodell sind unterhalb dieses neu hinzugefügten Knotens zu finden (vgl. Abbildung 8.8, GoogleSearchService).

8.3.4 Beispiel zur Integration in einen Geschäftsprozess

Als Beispiel für die Modellierung eines einfachen, verteilten Geschäftsprozesses wurde jeweils innerhalb eines Prozessmodells ein Web Service mit den generierten Aktivitäten angesprochen. Als Beispiele wurden die Web Services von Google und Amazon verwendet. Zuerst wurden die Web Services mit dem „Web Service Import Wizard“ zur Nutzung im Process Modeler verfügbar gemacht. Im nächsten Schritt wurde der Prozess modelliert (siehe Abbildungen 8.9 und 8.10). Dabei werden über eine Page Activity (in den jeweiligen Prozessmodellen durch große Rechtecke dargestellt) zuerst Informationen vom Nutzer abgefragt (Webseite siehe Abbildung 8.11), anschließend ausgelesen und als Parameter im Prozesskontext gespeichert, damit die jeweilige Web Service Aktivität (kleineres Rechteck in den Prozessmodellen) darauf zugreifen kann.[5] Die Ausgabe

[5] Bei der Modellierung der Amazon *ItemSearch* wurde die Nutzereingabe aus der Page Activity aus dem Web-Formular in einer Formbean gespeichert. Die Formularfelder werden automatisch auf die entsprechend benannten Variablen einer JavaBean gemappt, was mittels der durch eine Ellipse dargestellten *GetModel*-Aktivität durchgeführt wird.

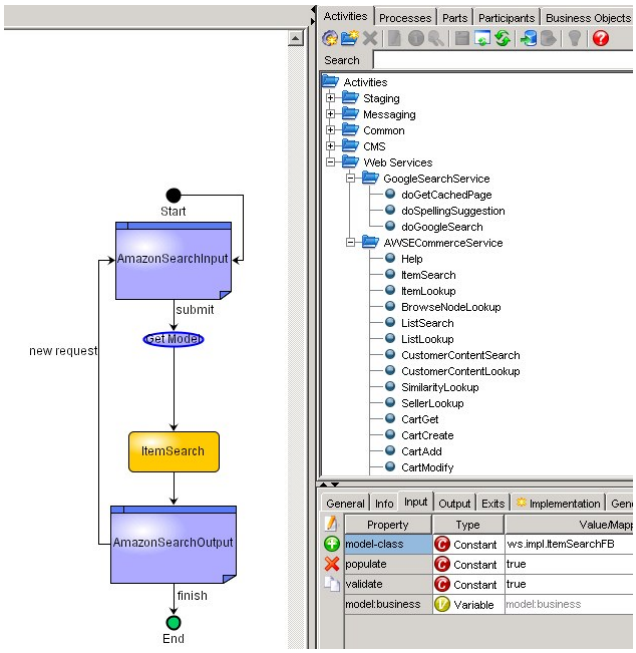

Abbildung 8.9: Prozessmodell zur Nutzung der ItemSearch des Amazon Web Services

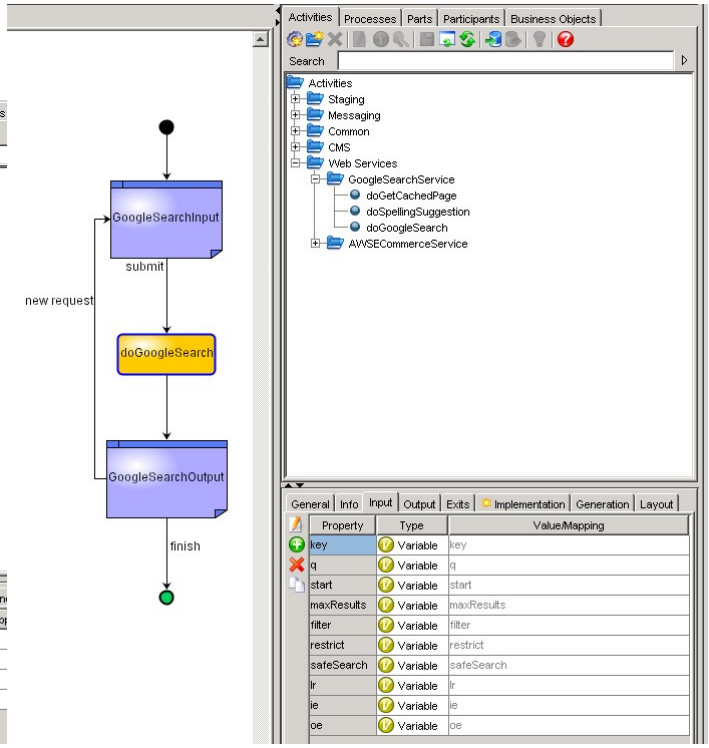

Abbildung 8.10: Prozessmodell zur Nutzung eines Google Web Services

erfolgt erneut über eine Page Activity, die mit einer JSP-Seite gestaltet wird. Die Ergebnis-Webseiten sind in den Abbildungen 8.12 und 8.13 dargestellt.

8.4 Anmerkungen zur Implementierung – Problembereiche

In den folgenden Abschnitten werden Besonderheiten und Herausforderungen beschrieben, die im Zuge der Implementierung des „Web Service Wizards“ aufgetreten sind.

8.4.1 Encoding Styles eines WSDL-Dokuments und Datentypen-Ermittlung

Zum Einlesen und als Zwischenrepräsentation eines WSDL-Dokuments wird die Klasse `WSDLInformation` benutzt. Das Parsen der WSDL-Datei erfolgt mit WSDL4J. Um die Ein- und Ausgabeparameter und den jeweiligen Datentyp einer Operation zu ermitteln, muss für jede Operation das zugehörige `<Message>`-Element und dessen einzelne `<Parts>`-Definitionen ausgewertet werden. Dabei treten Unterschiede auf je nachdem, ob es sich um einen document/literal oder RPC/encoded Dienst handelt, was im Folgenden näher erläutert wird.

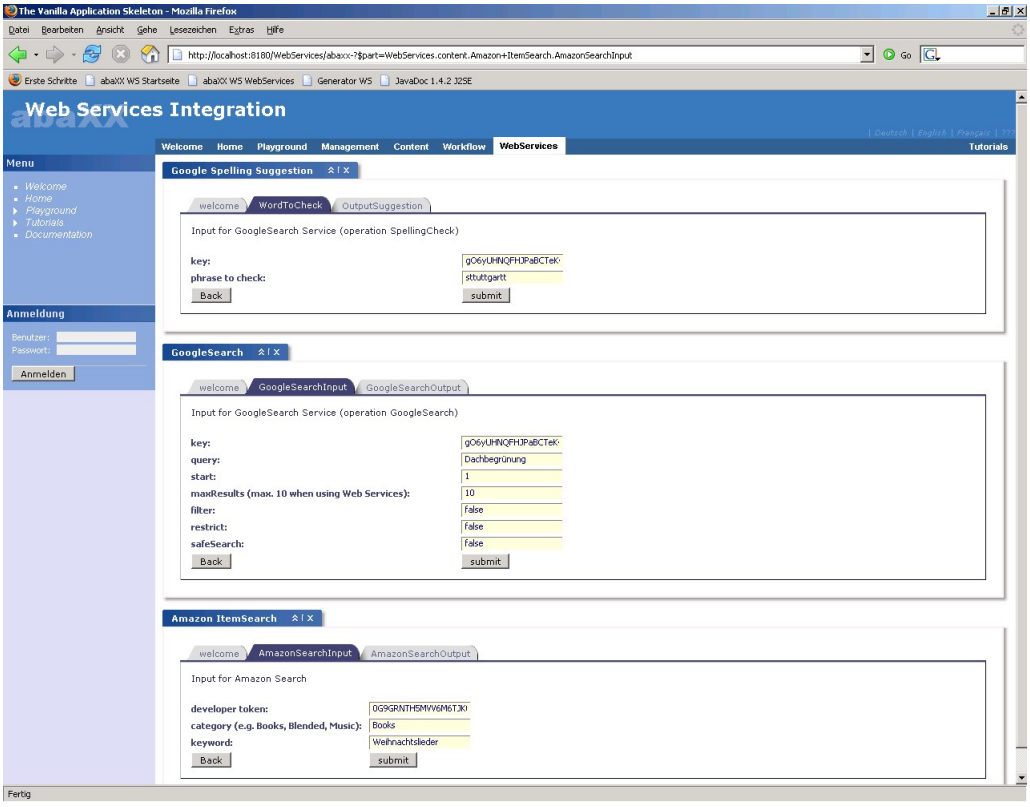

Abbildung 8.11: Beispiel zur Nutzung von Web Services – Eingabe von Daten durch den Nutzer

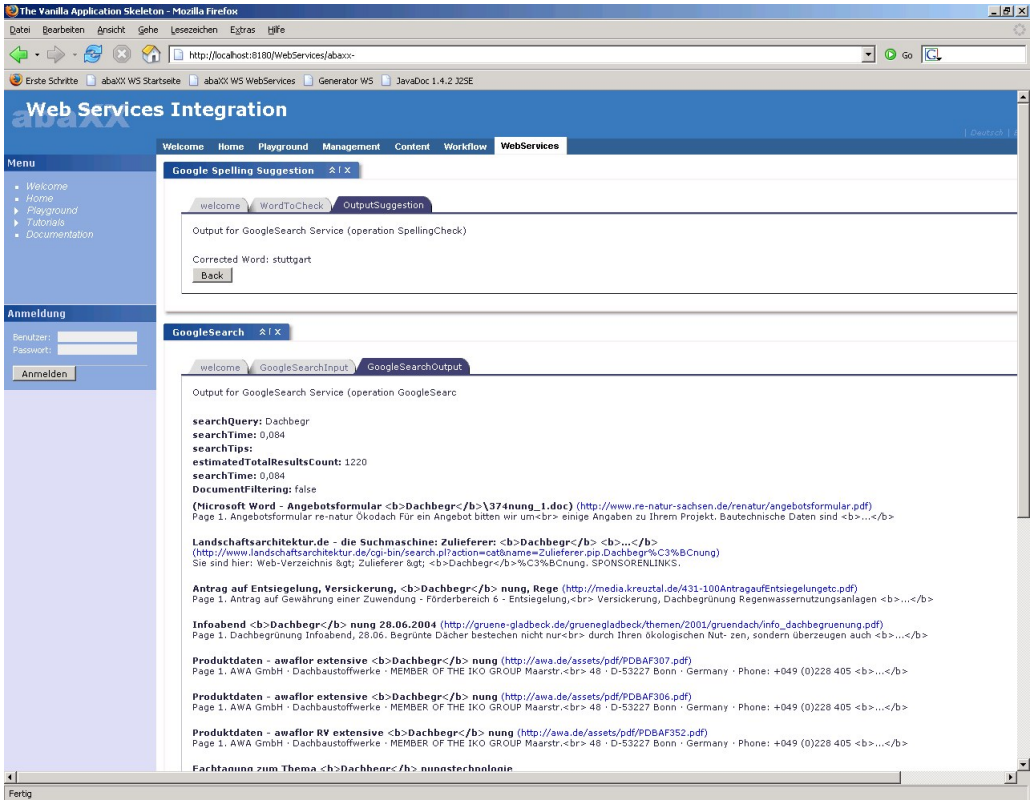

Abbildung 8.12: Beispiel zur Nutzung von Web Services – Anzeigen der Ergebnisse I.

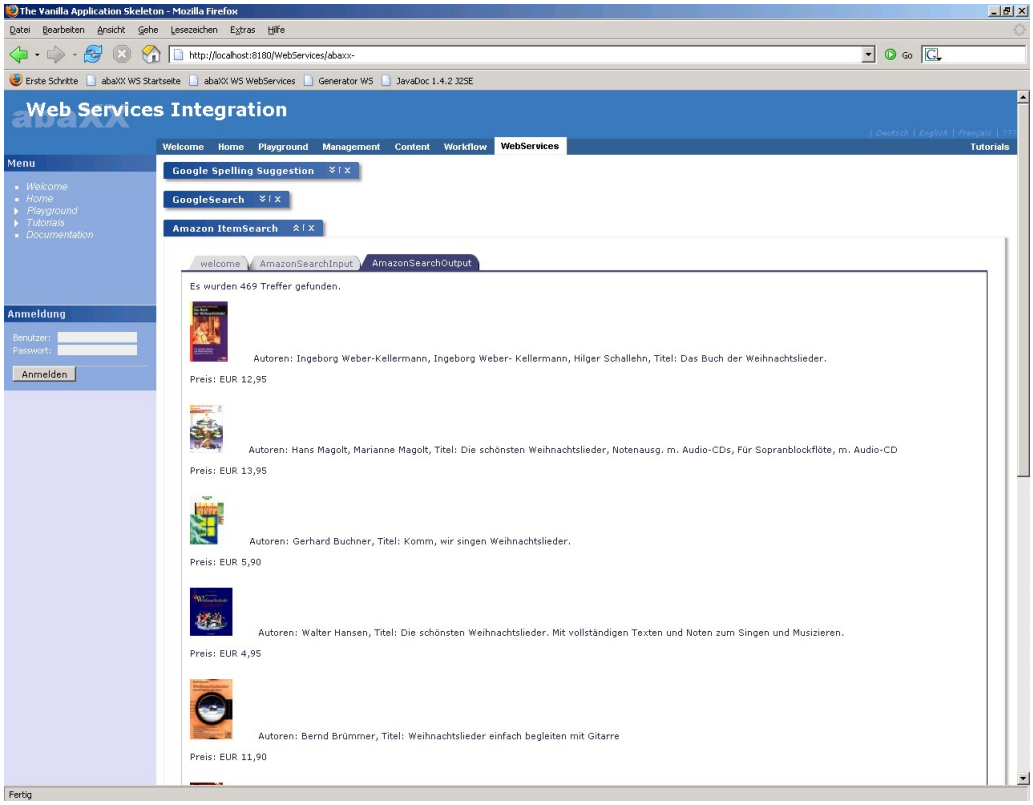

Abbildung 8.13: Beispiel zur Nutzung von Web Services – Anzeigen der Ergebnisse II.

Der GoogleSearchService verwendet als Encoding style RPC/encoded, was zu folgender Gestaltung seiner `<Message>`-Definition führt:

```
<message name="doGetCachedPage">
  <part name="key"   type="xsd:string"/>
  <part name="url"   type="xsd:string"/>
</message>
```

Als Encoding style des Web Services von Amazon wurde hingegen document/literal gewählt, was die `<Message>`-Definition folgendermaßen aussehen lässt:

```
<message name="ItemSearchRequestMsg">
   <part name="body" element="tns:ItemSearch"/>
</message>
```

Man kann nun entweder mit der WSDL4J-Methode `getElement()` oder `getType()` den Datentyp auslesen. Jedoch gibt es keine „neutrale“, vom Encoding style unabhängige Möglichkeit, den Datentyp zu ermitteln. Dies musste bei der Implementierung berücksichtigt werden.

Um die korrekten Datentypen für die Ein- bzw. Rückgabewerte eines Web Service-Aufrufs für die Activity-Implementation zu erhalten, werden die Angaben aus dem WSDL-Dokument mit Hilfe von AXIS überprüft. Die AXIS-Klasse `TypeMappingRegistry` beinhaltet für fast alle XML Schema-Datentypen Hilfsklassen zum (De-)Serialisieren von Objekten.

Bei der Überprüfung wird getestet, ob eine AXIS-Hilfsklasse für den aktuellen Message-Part-Datentyp vorhanden ist. Wenn ja, wird zusätzlich abgefragt, ob es sich um einen primitiven Datentyp oder ein Array handelt. Liefert die Überprüfung keinen Datentyp zurück, handelt es sich um einen komplexen XML Schema Datentyp, für dessen Zugriff von AXIS eine JavaBean erstellt wurde.

8.4.2 Variablen-Mapping zwischen Web Services und Prozessmodell

Ein Problembereich konzeptueller Natur betraf v.a. die Möglichkeit, Variablen aus dem Prozessmodell (d.h. Prozesskontext) für Web Service-Calls zu nutzen und Antwortnachrichten eines Web Services wieder in das Prozessmodell zu übernehmen. Innerhalb des Prozessmodells wird ausschließlich mit Java-Objekten, meist in Form von JavaBeans gearbeitet. Zum Austausch von Daten via Web Services werden aus Gründen der Interoperabilität Daten basierend auf einer XML Schema-Codierung übertragen. Die Schnittstelle, um von der XML-Welt in die Welt der Java-Objekte und umgekehrt zu wechseln, wird mit den Begriffen *Mapping* oder *„XML Data Binding“* bezeichnet und wird in diesem Fall durch die (De-)Serialisierungs-Funktionalitäten des Web Service Frameworks AXIS 1.3 realisiert.

Das Web Service Framework AXIS (1.x) erzeugt bei komplexen Datentypen JavaBeans auf Basis der XML Schema Definition eines WSDL-Dokuments. Diese Beans werden dazu genützt, zusammen mit Java-Basisdatentypen einen Web Service-Call zu tätigen. Für den Java-Programmierer scheint dies eine transparente Lösung darzustellen, so dass er von dem „Umweg“ über XML bei der Anwendung des Frameworks nichts mitbekommt und ausschließlich mit Java-Objekten arbeitet. Dieses Vorgehen wird auch im Zuge der Implementierung der Custom Activities gewählt.

Eine andere Möglichkeit, den Datenfluss zu gestalten, würde in einer durchgängigen Nutzung von XML-Datentypen bestehen. Im vorliegenden Prozessmodell hätte dies jedoch bedeutet, sämtliche POJOs (Plain Old Java Objects) des Prozessmodells in XML-Datentypen umzuwandeln und die Verarbeitung durch Web Services darauf basieren zu lassen. Besonders hinsichtlich einer Modellierung von Geschäftsprozessen mittels WS-BPEL wäre eine solche Option wünschenswert, da der Datenfluss über XML-Daten ausgeführt wird. In der Konzeptionsphase wurden beispielsweise Überlegungen angestellt, dieses Vorgehen zu implementieren. Werkzeuge zur Serialisierung von (auch komplexen) Java-Objekten in XML sind zwar vorhanden (z.B. XStream[6]). Jedoch wäre zur Weiterverarbeitung das automatische Erstellen eines XML Schemas wünschenswert, wofür leider bei der Recherche im Zuge dieser Arbeit keine Tool-Unterstützung im Open-Source-Bereich gefunden wurde.

XML-Daten müssen zur Verwendung mit Java-Objekten immer (de-)serialisiert werden. Aus diesem Grund wäre eine „nahtlose“ Integration von XML-Datentypen in die jewei-

[6] `http://xstream.codehouse.org`, verifiziert am 09.02.2006, 15:08 MEZ

ligen Programmiersprachen wünschenswert: Es könnten dann beispielsweise XML Schema Datentypen direkt genutzt werden, d.h. im Idealfall so wie native Datentypen einer Programmiersprache. Lämmel und Meijer (2005) beschreiben in ihrem Artikel die Problematik des Mappings in genereller Weise. Sie kommen zum Schluss, dass bislang eine Vielzahl von Praxislösungen entwickelt worden ist, jedoch eine systematische, wissenschaftliche Auseinandersetzung mit diesem Themenbereich noch nicht stattgefunden hat.

Häufig wird das „Wandeln zwischen den Welten“ (XML und Objekte) als sehr inperformant und ineffizient beschrieben, was auch auf die Nutzung von Web Services allgemein übertragen wird. Tatsächlich fällt die Informationsdichte in textbasierten XML-Dokumenten, verglichen mit binären Daten, geringer aus. Jedoch müssen Web Services aus diesem Grund nicht automatisch inperformanter sein. Dostal et al. (2005, 131ff.) haben dazu in praktischen Experimenten Web Service Calls (via AXIS) mit Java RMI und CORBA verglichen und gelangen zu folgenden Ergebnissen (Dostal et al. 2005, 161f.):

- Die Netzwerklast und Parameter wie die Größe der Pakete üben einen extremen Einfluss aus. Allein schon durch Optimierungen auf Netzwerkebene besteht ein großes Verbesserungspotential.
- Das Parsen von XML-Dateien ist ein zu vernachlässigender Faktor. In manchen Testfällen wäre SOAP besser als RMI und CORBA, manchmal umgekehrt, ohne dass es eine generelle Tendenz zu beobachten ist.
- Das HTTP-Protokoll zum Transport der SOAP-Nachrichten ist in manchen Messreihen der entscheidende Flaschenhals gewesen. Daher lohnt sich eine Überprüfung des verwendeten Transportprotokolls auf seine Zweckmäßigkeit.

8.4.3 Blockierende Operationen

Werden Web Service-Calls mittels des Frameworks AXIS 1.3 realisiert, wie im Prototyp des „Web Service Import Wizards“ geschehen, sind dies ausschließlich sog. „blocking calls“. D.h., die Anwendung sendet die Anfrage an den Web Service und blockiert solange, bis eine Antwort vom angesprochenen Web Service eintrifft. Das verwendete Framework unterstützt ausschließlich diese Art der Request-Response-Verarbeitung, bei der eine direkte Antwort auf eine Anfrage erwartet wird (vgl. Gegenüberstellung zu AXIS 1.x in Kapitel 7.2.2 auf Seite 67f.).

Dies kann sich nachteilig auf das Antwortverhalten der Portallösung auswirken: Der Nutzer nimmt im Webbrowser einen verzögerten Seitenaufbau wahr, ohne dass er Rückmeldung z.B. über eine Fortschrittsanzeige erhält. Sollte ein Web Service gar nicht mehr in der Lage sein, eine Antwort zu versenden, z.B. auf Grund eines Fehlers auf Serverseite, würde der Webbrowser nach langem Warten mit einem Time-Out Fehler abbrechen. Zudem ist, falls dem Nutzer überhaupt die Möglichkeit dazu eingeräumt werden soll, ein Abbrechen dieser Request-Response-Verarbeitung nicht möglich. Eine Umgehung dieses

Problems bestünde darin, den Web Service-Call als separaten Thread zu implementieren, was jedoch Schwierigkeiten bei der Synchronisation von Daten und dem Zugriff auf den Prozesskontext bereiten würde.

8.5 Zusammenfassung

Im Rahmen dieser Abschlussarbeit wurde ein Prototyp eines „Web Service Import Wizards“ zur Nutzung von Web Services in der Prozessportal-Lösung der Firma abaXX Technology erstellt. Die praktische Umsetzung und das dafür gewählte Vorgehen sind Gegenstand dieses Kapitels. Nachdem die Anforderungen durch Anwendungsfälle konkretisiert wurden, musste ein geeignetes Web Service Framework ausgewählt werden. Die Wahl fiel auf Apache AXIS 1.3, da es das momentan stabilste Web Service Framework ist, das zugleich über die meisten (teilw. prototypischen) Erweiterungsmöglichkeiten verfügt. Während der Implementierung wurden als besondere Problembereiche der Encoding style und die Ermittlung von Datentypen aus einem WSDL-Dokument sowie das Variablen-Mapping zwischen Prozessmodell und den bei Web Services genutzten Datentypen identifiziert.

9 Fazit und Ausblick

Die Ergebnisse dieser Arbeit werden in diesem Kapitel zusammengefasst und mögliche Erweiterungen des Prototyps präsentiert. Abschließend wird ein Ausblick auf künftige Entwicklungen durch die Kombination von Web Services und Technologien des Semantic Webs geliefert.

9.1 Erweiterungen der Prozessportal-Lösung von abaXX Technology um zukünftige Web Service-Komponenten

In der vorliegenden Arbeit wurde gezeigt, wie Web Services in die Prozessportal-Lösung der Firma abaXX Technology integriert und nutzbar gemacht werden können. Dies wurde über eine prototypische Software, den „Web Service Import Wizard“, ermöglicht, der eine clientseitige Nutzung von Web Services innerhalb eines Prozessmodells unterstützt.

Für einen unternehmensübergreifenden Einsatz von Web Services, der über die Portal-Lösung von abaXX Technology abgewickelt werden soll, ist eine Unterstützung von Sicherheitsmerkmalen und Transaktionen unabdingbar. Diese beiden Funktionalitäten sind die wichtigsten Aspekte, die es bei einer Weiterentwicklung zu beachten gilt. Außerdem empfiehlt es sich, den „Web Service Import Wizard“ zu einem allgemeinen „Web Service Wizard“ zu erweitern: Web Services sollen nicht nur auf Seiten des Clients konsumiert werden, sondern es sollen zukünftig auch Teile eines Prozessmodells als Dienst angeboten werden. Dazu müsste eine serverseitige Komponente hinzugefügt werden.

Eine weitere Möglichkeit, Web Services in die Prozessportal-Lösung zu integrieren besteht in der Unterstützung des Standards „Web Services for Remote Portlets“[1] (WSRP). Hierbei liefert ein Web Service bereits zur Darstellung formatierte Daten z.B. in HTML, die direkt als Portlet dargestellt werden können. Bei dieser Herangehensweise liegt der Schwerpunkt auf der direkten Interaktion des Nutzers mit einem Web Service über die Nutzungsoberfläche. Die im Rahmen dieser Arbeit gewählte Herangehensweise ermöglicht dagegen die Nutzung von Web Services innerhalb eines Prozessmodells. Web Services können dort als Datenquelle verwendet und ihre Ergebnisse weiterverarbeitet werden, wobei die Darstellung von Ergebnissen im Portal Aufgabe des Programmierers ist. Eine Unterstützung des Standards WSRP ist eine zukünftige Erweiterungsmöglichkeit für die Portallösung.

[1] Spezifikation unter `http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf` in der Version 1.0, verifiziert am 07.03.2006

9.2 Ausblick: Web Services zur Anwendungsintegration

Eine Service-oriented Architecture kann durch Web Services umgesetzt werden. Dabei können weiterführende Konzepte im Bereich „Web Services“ Schritt für Schritt ergänzt werden, z.B. durch Hinzufügen von Komponenten für eine sichere Übertragung oder Transaktionsabwicklung. Insgesamt kann eine SOA in kleinen, aufeinander folgenden Schritten eingeführt werden, ohne dass beispielsweise von einem Tag auf den anderen eine Anwendungslandschaft durch ein neues, monolithisches Software-System abgelöst werden muss.

Der große Vorteil einer SOA liegt darin, dass sie ein geeignetes Mittel darstellt, Geschäftsabläufe und -prozesse in Zusammenarbeit mit Experten aus verschiedenen Fachrichtungen miteinander zu koordinieren. Die Fachbereiche formulieren ihre Sicht der Prozesse und die dazu benötigten Diensterbringer, die bei der Bearbeitung eines Prozesses tätig werden. Diese Gliederung kann bei der EDV-technischen Umsetzung der einzelnen Dienste aufgegriffen werden und ermöglicht somit eine große Nähe zwischen Fachkonzept und EDV-Konzept. Dies kommt z.B. Auftraggebern und Software-Dienstleistern bei der Spezifikation in der Planungsphase zu Gute, da eine annähernd gemeinsame Sprache zur Modellierung von Prozessen mittels Diensten gefunden werden kann und somit Reibungsverluste vermieden werden.

Web Services werden vor allem über Unternehmensgrenzen hinweg starke Verbreitung finden, um Anwendungen miteinander zu koppeln. Dies basiert vor allem auf ihrer (relativen) Einfachheit, den Standardisierungbestrebungen und der Interoperabilität, die eine breite Nutzung erleichtert. Aus software-architektonischer Sicht sind Web Services die idealen Bausteine zur Anwendungsintegration.

9.3 Ausblick: Semantic Web und Semantic Web Services

Zum jetzigen Zeitpunkt erfordern sämtliche Lösungen mit Web Services Programmierarbeit. Menschlicher Sachverstand und Intelligenz zur Auswahl und zum Zusammenfügen der einzelnen Dienste ist dabei unerlässlich. Die Erschließung der Bedeutung eines Dienstes, d.h. dessen Semantik, bleibt dem Menschen vorbehalten. Zukünftiges Ziel, das eine weitere Automatisierung verspricht, ist die Verknüpfung von Web Services mit Konzepten des Semantic Webs, um letztlich „Semantic Web Services“ zu erhalten.

Im Semantic Web sollen Informationen nicht nur für den Menschen verständlich, sondern auch durch einen Computer erschließbar sein, um auf Grund formaler Beschreibungselemente eine Bedeutung zu inferieren. Beispielsweise kann so unterschieden werden, dass es sich bei einer fünfstelligen Zahl entweder um eine deutsche Postleitzahl

oder um die Artikelnummer eines Online-Shops handelt. Oder eine Web-Seite beinhaltet Informationen zum Thema Fußball, so dass automatisch auf die zugehörige Oberkategorie „Sport“ geschlossen werden kann, ohne dass dafür menschliches Zutun erforderlich ist. Ein anderes Beispiel ist die Auswahl eines Web Services, der Echtzeit-Börseninformationen liefert. Dabei stellen sich bei der Auswahlentscheidung für den Computer folgende Fragen: Wie heißt der Dienst (z.B. StockQuoteService), was ist mit dem Konzept „Echtzeit“ verbunden, wie kann der Dienst eingesetzt werden (was z.B. in einem WSDL-Dokument beschrieben ist) und zu welchen Bedingungen und Kosten kann ein Dienst eingesetzt werden (d.h., dass das Konzept Abrechnungsmodell und das Konzept der Allgemeinen Geschäftsbedingungen von einem Computer „verstanden“ werden muss)? Die bislang im Web Service-Umfeld eingesetzten Technologien weisen keine semantischen Komponenten auf, auch nicht das zur Dienstsuche vorgestellte UDDI (siehe Kapitel 4.3) (vgl. Dostal et al. 2005, 376ff.).

Zur Realisierung des Semantic Webs werden vom W3C u.a. folgende Standards gepflegt: Zum einen ist dies das *Resource Description Framework* (RDF) zur standardisierten Beschreibung einer Ressource und deren Metadaten auf XML-Basis. Zum anderen geschieht dies mittels der *Web Ontology Language* (OWL), um Beziehungen zwischen Ressourcen auf Basis einer Ontologie zu formulieren. Eine Ontologie ist eine formale Beschreibung eines bestimmten Gegenstandsbereichs (einer sog. Domäne) mittels Konzepten und den Beziehungen zwischen den Konzepten.

Um die Konzepte des Semantic Webs mit denen von Web Services zu sog. „Semantic Web Services“ zu kombinieren, werden momentan Forschungsprojekte vorangetrieben. Die bislang erzielten Ergebnisse sind jedoch noch sehr weit entfernt von einem Reifegrad und einer Verbreitung, wie dies bei den Standard-Web Service Spezifikationen (SOAP, WSDL, UDDI) der Fall ist. Einen aktuellen Überblick über die Problematik und Lösungsansätze liefern Polleres et al. (2006) in ihrem Artikel.

Mittels Web Services bieten sich vielfältige Möglichkeiten, Anwendungen miteinander zu integrieren. Die Integration in bestehende Konzepte und Kopplung mit vorhandenen Technologien führt in Zukunft zu weiteren Einsatzmöglichkeiten der Technologie Web Services.

Literaturverzeichnis

[abaXX 2005] TECHNOLOGY AG abaXX: *Dokumentation zur Workflow-Engine der abaXX.components Version 4.6*. 2005

[Alonso et al. 2004] ALONSO, Gustavo ; CASATI, Fabio ; KUNO, Harumi ; VIJAY, Machiraju: *Web Services. Concepts, Architectures and Applications.* Berlin, Heidelberg : Springer-Verlag, 2004

[Apfel 2004] APFEL, Steffen: *Ein generisches Framework zur grafischen Anbindung von Web-Services*, Universität Kaiserslautern. AG Integrierte Kommunikationssysteme., Diplomarbeit, 2004. – URL http://www.icsy.de/~archiv/DPArchiv.0136.pdf. – Zugriffsdatum: 18.01.2006, 17:33 MEZ

[AxisDoc 2005] N.N.: *Axis Architecture Guide. Version 1.2*. 2005. – URL http://ws.apache.org/axis/java/index.html. – Zugriffsdatum: 31.08.2005, 10:56 MEZ

[Ballinger et al. 2004] BALLINGER, Keith ; EHNEBUSKE, David ; FERRIS, Christopher ; GUDGIN, Martin ; LIU, Canyang K. ; NOTTINGHAM, Mark ; YENDLURI, Prasad: *Basic Profile Version 1.1*. 2004. – URL http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html. – Zugriffsdatum: 11.01.2006, 11:54 MEZ

[Bierkoch 2005] BIERKOCH, Martin: *Webservice-Entwicklung für eine J2EE-Anwendung*, Fachhochschule für Technik Esslingen. Fachbereich Informationstechnik, Diplomarbeit, 2005

[Buhler et al. 2004] BUHLER, Paul A. ; STARR, Christopher ; SCHRODER, William H. ; VIDAL, José M.: *Preparing for Service-Oriented Computing: a composite design pattern for stubless Web service invocation*. 2004. – URL http://citeseer.ist.psu.edu/buhler04preparing.html. – Zugriffsdatum: 10.02.2006, 15:55 MEZ

[Burghardt 2004] BURGHARDT, Markus: *Web Services. Aspekte von Sicherheit, Transaktionalität, Abrechnung und Workflow.* Wiesbaden : Deutscher Universitäts-Verlag, 2004

[Butek 2005] BUTEK, Russel: *Which style of WSDL should I use?* 2005. – URL http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/. – Zugriffsdatum: 11.01.2006, 10:38 MEZ

[Dostal et al. 2005] DOSTAL, Wolfgang ; JECKLE, Mario ; MELZER, Ingo ; ZENGLER, Barbara: *Service-orientierte Architekturen mit Web Services. Konzepte – Standards – Praxis*. 1. Auflage. München : Spektrum Akademischer Verlag, Elsevier GmbH, 2005

[Duden 2001] *Duden. Herkunftswörterbuch. Etymologie der deutschen Sprache.* 3., völlig neu bearbeitete und erweiterte Auflage. Mannheim, Leipzig, Wien, Zürich : Dudenverlag, 2001

[Duftler et al. 2001] DUFTLER, Matthew J. ; MUKHI, Nirmal K. ; SLOMINSKI, Aleksander ; WEERAWARANA, Sanjiva: *Web Services Invocation Framework (WSIF)*. 2001. – URL http://citeseer.ist.psu.edu/duftler01web.html. – Zugriffsdatum: 10.02.2006, 15:58 MEZ

[Eberhart und Fischer 2004] EBERHART, Andreas ; FISCHER, Stefan: *Web Services. Grundlagen und praktische Umsetzung mit J2EE und .NET*. Stuttgart : Hanser Verlag, 2004. – URL http://files.hanser.de/hanser/docs/20040401_2445154312-31966_3-446-22530-7.pdf. – Zugriffsdatum: 22.08.2005, 11:15 MEZ. – Leseprobe Kapitel 10: Werkzeugunterstützung für SOAP und WSDL

[Fremantle 2002] FREMANTLE, Paul: *Applying the Web services invocation framework. Calling services independent of protocols.* 2002. – URL http://www-128.ibm.com/developerworks/webservices/library/ws-appwsif.html?open\&l=889%2Ct=gr. – Zugriffsdatum: 05.02.2006, 15:30 MEZ

[Frotscher 2005] FROTSCHER, Thilo: Web Services der dritten Generation. 3 Axis2-Kernentwickler im Gespräch. In: *Java Magazin* (2005), Nr. 10. – URL http://www.javamagazin.de/itr/online_artikel/psecom,id,761,nodeid,11.html. – Zugriffsdatum: 05.10.2005, 09:56 MEZ

[Gartner 2002] MCCOY, David W. ; MORELLO, Diane ; MIKLOVIC, Dan ; EARLEY, Annemarie ; NICOLETT, Mark ; FULTON, Roger ; STONE, Lisa: *Gartner Predicts 2002: Top 10 Predictions*. 2002. – URL http://www.gartner.com/DisplayDocument?doc_cd=103726. – Zugriffsdatum: 20.02.2006, 14:26 MEZ

[König 2005] KÖNIG, Dieter: *Web Services Business Process Execution Language (WS-BPEL)*. 2005. – URL http://www.oasis-open.org/committees/download.php/15424/OASIS%20WS-BPEL%202.0.ppt. – Präsentation bei BuilConn am 09.11.2005 in Amsterdam

[Lämmel und Meijer 2005] LÄMMEL, Ralf ; MEIJER, Erik: Mappings make data processing go'round. (2005). – URL http://homepages.cwi.nl/~ralf/mappings/paper.pdf. – Zugriffsdatum: 10.02.2006, 15:40 MEZ. – Version October 29, 2005. A prior version appeared in the pre-proceedings of GTTSE 2005

[Lorenzelli-Scholz 2005] LORENZELLI-SCHOLZ, Domenico: Service-orientierte Integration mit einem „Enterprise Service Bus". In: *Java Spektrum* (2005), Nr. 10. – URL http://www.sigs.de/publications/os/2005/06/lorenzelli_scholz_OS_06_05.pdf. – Zugriffsdatum: 05.10.2005, 09:56 MEZ

[Marks 2005] MARKS, Michael: *Modellierung und Automatisierung von Web Service Resourcen (sic!) am Beispiel von BPEL Prozessen*, Universität Stuttgart. Institut für Architektur von Anwendungssystemen (IAAS), Diplomarbeit, 2005. – URL ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/DIP-2308/DIP-2308.pdf. – Zugriffsdatum: 18.01.2006, 17:31 MEZ

[Martens 2004] MARTENS, Axel: *Verteilte Geschäftsprozesse. Modellierung und Verifikation mit Hilfe von Web Services.* Stuttgart und Berlin : WiKu-Verlag, 2004

[zur Muehlen et al. 2004] MUEHLEN, Michael zur ; NICKERSON, Jeffrey V. ; SWENSON, Keith D.: Developing Web Services Choreography Standards – The Case of REST vs. SOAP. In: *Decision Support Systems* (2004), Nr. 37. – URL http://www.stevens-tech.edu/jnickerson/MIZU.JENI.KESW-DSS.pdf. – Zugriffsdatum: 22.08.2005, 11:15 MEZ. – to appear

[Ort 2004] ORT, Ed: Service-Oriented Architecture and Web Services: Concepts, Technologies, and Tools. / Sun Microsystems. URL http://java.sun.com/developer/technicalArticles/WebServices/soa2/soa2.pdf. – Zugriffsdatum: 06.10.2004, 10:42 MEZ, 2004. – Forschungsbericht

[Perera 2005] PERERA, Srinath: *Introducing Axis 2, the Next Generation of the Apache Web Service Stack*. 2005. – URL http://www.developer.com/java/ent/article.php/3525481. – Zugriffsdatum: 31.08.2005, 10:56 MEZ

[Perera und Ranabahu 2005] PERERA, Srinath ; RANABAHU, Ajith: *Axis2 - The Future of Web Services*. 2005. – URL http://entwickler.com/itr/online_artikel/show.php3?nodeid=97\&id=747. – Zugriffsdatum: 30.08.2005, 16:50 MEZ

[Polleres et al. 2006] POLLERES, Axel ; LAUSEN, Holger ; LARA, Rubén: *Semantische Beschreibung von Web Services*. In: PELLEGRINI, Tassilo (Hrsg.) ; BLUMAUER, Andreas (Hrsg.): *Semantic Web – Wege zur vernetzten Wissensgesellschaft*. Hildesheim, Zürich, New York : Springer, 2006. – URL http://www.polleres.net/publications/poll-etal-2006.pdf. – Zugriffsdatum: 03.03.2006, 20:00 MEZ. – Preprint auf der Homepage des Autors

[Reichert und Stoll 2004] REICHERT, Manfred ; STOLL, Dietmar: Komposition, Choreographie und Orchestrierung von Web Services – Ein Überblick. In: *EMISA Forum* 24 (2004), Nr. 2, S. 21 – 32. – URL http://www.informatik.uni-ulm.de/dbis/01/dbis/downloads/ReSt04.pdf. – Zugriffsdatum: 17.01.2005, 17:46 MEZ

[Reichmayr 2003] REICHMAYR, Christian: *Collaboration und WebServices. Architekturen, Portale, Techniken und Beispiele*. Berlin, Heidelberg : Springer, 2003

[Rost 2003] ROST, Dirk: *Die Beschwörung. Crashkurs Apache Web Services Invocation Framework*. 2003. – URL http://www.javamagazin.de/itr/online_artikel/psecom,id,424,nodeid,11.html. – Zugriffsdatum: 15.08.2005, 09:56 MEZ

[Schmidt et al. 2005] SCHMIDT, Marc-Thomas ; HUTCHISON, Beth ; LAMBROS, Peter ; PHIPPEN, Rob: The Enterprise Service Bus: Making service-oriented architecture real. In: *IBM Systems Journal* 44 (2005), Nr. 4, S. 781–797. – URL http://www.research.ibm.com/journal/sj/444/schmidt.pdf. – Zugriffsdatum: 12.02.2006, 10:56 MEZ. – DOI: 10.1147/sj.444.0781

[Silberberger 2003] SILBERBERGER, Holger: *Collaborative Business und Web Services. Ein Managementleitfaden in Zeiten technologischen Wandels*. Berlin, Heidelberg : Springer, 2003

[Spall 2003] SPALL, Andreas: *Client-Entwicklung mit JFC oder SWT?* 2003. – URL http://www.oio.de/client-swing-swt.htm. – Zugriffsdatum: 10.02.2006, 15:36 MEZ

[Trenaman 2005] TRENAMAN, Adrian: *Using Open Source Software for SOA*. 2005. – URL http://objectwebcon06.objectweb.org/xwiki/bin/download/Main/DetailedSession/A-Trenaman-SOA.pdf. – Zugriffsdatum: 20.03.2006, 11:01 MEZ

[Vlachikis et al. 2005] VLACHIKIS, Joannis ; KIRCHHOF, Anja ; GURZKI, Thorsten ; SATH, Dieter (Hrsg.) ; HINDERER, Henning (Hrsg.): *Marktübersicht Portalsoftware 2005*. Stuttgart : Fraunhofer IRB Verlag, 2005. – URL http://www.ebi.iao.fraunhofer.de/docs/Marktuebersicht_Portalsoftware_2005_web.pdf. – Zugriffsdatum: 22.08.2005, 11:15 MEZ. – Kostenloser Auszug ohne detaillierte Produktbeschreibungen

[Weerawarana et al. 2005] WEERAWARANA, Sanjiva ; CURBERA, Francisco ; LEYMANN, Frank ; STOREY, Tony ; FERGUSON, Donald F.: *Web Services Platform Architecture. SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More.* Upper Saddle River, NJ et al. : Prentice Hall, 2005

[Wehner 2005] WEHNER, Heinz: *Was bringt ein Enterprise Service Bus?* 2005. – URL http://www.computerwoche.de/produkte_technik/software/554063/. – Zugriffsdatum: 12.02.2006, 15:36 MEZ

Eigenständigkeitserklärung

Ich erkläre, dass ich diese Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die diesen Quellen und Hilfsmitteln wörtlich oder sinngemäß entnommenen Ausführungen als solche kenntlich gemacht habe.

Burladingen, den 27. März 2006 *Joachim Pfister*